Research Product 93-04

Models of Morse Code Skill Acquisition: Simulation and Analysis

February 1993

Automated Instructional Systems Technical Area
Training Systems Research Division

U.S. Army Research Institute for the Behavioral and Social Sciences

U.S. ARMY RESEARCH INSTITUTE
FOR THE BEHAVIORAL AND SOCIAL SCIENCES

A Field Operating Agency Under the Jurisdiction
of the Deputy Chief of Staff for Personnel

EDGAR M. JOHNSON
Acting Director

Research accomplished under contract
for the Department of the Army

Battelle Memorial Institute, Inc.

Technical review by

Mark A. Sabol
Robert A. Wisher

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	1993, February	Final Jun 91 - Dec 91

4. TITLE AND SUBTITLE
Models of Morse Code Skill Acquisition: Simulation and Analysis

5. FUNDING NUMBERS
DAAL03-91-C-0034
62785A
791
3302
H1

6. AUTHOR(S)
Fisher, Donald L., University of Massachusetts, and
Townsend. James T., Indiana University

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Battelle Memorial Institute, Inc.
Research Triangle Park, NC

8. PERFORMING ORGANIZATION REPORT NUMBER
--

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Institute for the Behavioral and Social Sciences
ATTN: PERI-II
5001 Eisenhower Avenue
Alexandria, VA 22333-5600

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
ARI Research Product 93-04

11. SUPPLEMENTARY NOTES
Contracting Officer's Representative, Robert A. Wisher

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release;
distribution is unlimited.

12b. DISTRIBUTION CODE
--

13. ABSTRACT (Maximum 200 words)

The simulation described in this report predicts details of performance for students learning to copy Morse code at high speeds of transmission. Specifically, the model predicts the probability of a correct response, an incorrect response, a period (no guess) response, no response, and a correct response for each of the five serial positions in a group. The simulation also predicts the time it takes to execute both a correct and incorrect response and the time it takes to execute a period response. The model was derived from a cognitive analysis of the information processing demands on students and modeled with order-of-processing diagrams. A preliminary test was conducted with response data from students at the U.S. Army Intelligence School, Fort Devens, Massachusetts.

14. SUBJECT TERMS

Skill training	Cognitive models
Order of processing	Short term memory
Information processing	Morse code

15. NUMBER OF PAGES
110

16. PRICE CODE
--

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

Research Product 93-04

Models of Morse Code Skill Acquisition: Simulation and Analysis

Donald L. Fisher
University of Massachusetts

James T. Townsend
Indiana University

Automated Instructional Systems Technical Area
Robert J. Seidel, Chief

Training Systems Research Division
Jack H. Hiller, Director

U.S. Army Research Institute for the Behavioral and Social Sciences
5001 Eisenhower Avenue, Alexandria, Virginia 22333-5600

Office, Deputy Chief of Staff for Personnel
Department of the Army

February 1993

Army Project Number
2Q162785A791

Manpower, Personnel, and Training

FOREWORD

To ensure that the U.S. Army's soldiers acquire the skills and knowledge necessary to perform their jobs successfully, the U.S. Army Research Institute for the Behavioral and Social Sciences (ARI) performs behavioral research to develop methods of training that can improve skill acquisition. Morse code copy training is one skill area that continues to challenge the research community to identify training strategies that can alter the training attrition pattern, particularly during the speed building phase.

This report describes a quantitative model that simulates the perceptual-motor skill responsible for successful Morse code copy. This model will allow detailed characteristics in the underlying human information processing mechanics to be simulated and compared to actual performance. Further application of this model might enable the "at-risk" students to be identified early in the speed building phase of training and thus designated for specialized training. In general, this research adds to our understanding of the acquisition of skilled performance.

EDGAR M. JOHNSON
Acting Director

MODELS OF MORSE CODE SKILL ACQUISITION: SIMULATION AND ANALYSIS

EXECUTIVE SUMMARY

Requirement:

The U.S. Army Intelligence School at Fort Devens (USAISD) has been experiencing an unacceptably high rate of attrition. (The attrition rate was recently reduced after the introduction of a new training device.) The Assistant Deputy Chief of Staff for Training, HQ Training and Doctrine Command (TRADOC), requested that the U.S. Army Research Institute for the Behavioral and Social Sciences (ARI) examine the problem and pursue a course of research that could lead to a reduction in the attrition rate. The simulation and analysis of performance and models to differentiate between successful students and attrites were important steps in the approach.

Procedure:

In a briefing to HQ TRADOC, a cognitive process model of the Morse copy skill was proposed as a means to understand the intricacies of the skill. An elaboration of this model was briefed to USAISD staff. Through a contractual arrangement with the U.S. Army Research Office, researchers developed simulation models that applied tools in queuing networks and order-of-processing diagrams to the task of a student learning to copy Morse code at progressively faster speeds.

Findings:

A simulation model was derived from a cognitive analysis of the information processing demands on students and modeled with order-of-processing diagrams. Various assumptions about the "copy behind" phenomenon--not responding until a subsequent character is presented--were made. The model predicts the probability of a correct response, an incorrect response, a period (no guess) response, no response, and a correct response for each of the five serial positions in a group. The simulation also predicts the time it takes to execute both a correct and incorrect response and the time it takes to execute a period response. A preliminary test was conducted with response data from students at the U.S. Army Intelligence School, Fort Devens, early in the speed building phase of training. The models fit the observed data quite well, with only the predicted response times for incorrect responses much different from the observation.

Utilization of Findings:

These models will be tested with data from students who either acquired the skill rapidly or attrited during the speed building phase. This will establish whether response patterns early in the speed building phase can distinguish the quick learners from the likely attrites. These findings, in turn, will be presented to USAISD for consideration in identifying students for specialized training.

MODELS OF MORSE CODE SKILL ACQUISITION: SIMULATION AND ANALYSIS

CONTENTS

MODELS OF MORSE CODE SKILL ACQUISITION: SIMULATION AND ANALYSIS

INTRODUCTION

Background

The U.S. Army Intelligence School at Fort Devens, Massachusetts, trains Morse code copying to about 1,000 individuals per year from the four armed services. The goal is to train military personnel to copy Morse characters at the rate of 20 groups per minute (gpm; 1 group equals 5 characters) with 96 percent accuracy. The training has historically had a high rate of attrition, at times over 30 percent (DOES, 1990). For those who succeed in the basic Morse course, the training time has a wide degree of individual variability, ranging from 200 to 1180 hours to achieve proficiency. By far, most training time is spent in the speed building phase where students try to progress incrementally from 6 gpm to 20 gpm. Since attrition is most likely to occur during speed building and well into the training program, its effect can be costly.

Although the learning of Morse code has long been a topic of research in psychology (Bryan and Harter, 1897), the high training attrition persists. The learning of individual Morse characters is fairly straightforward, but the ability to copy groups of characters accurately at increasing rates of transmission is difficult to master for many individuals, even after hundreds of hours of practice. It is this difficulty in being unable to progress to a higher speed, not the lack of motivation or practice, that leads to the costly attrition.

The copy task requires listening to a sequence of Morse code characters (each character is itself a sequence of dahs and dits) and responding with the corresponding character on a standard keyboard. The ability to copy Morse code characters during rapid rates of transmission (100 characters per minute) is clearly demanding. To copy at speeds approaching 20 gpm burdens the underlying cognitive processes, pushing relevant memories to their limits. It requires the simultaneous processing of different stimuli, frequently responding to one character while another is being perceived. Perhaps the memory capacity of some operators is exceeded, or perhaps the concurrent processing

abilities of other operators are pushed too far. Present-day computer technology and theoretical frameworks for modeling cognitive processes suggest that a closer study can be made of the development of proficiency during speed building than was previously possible. The results of this more extensive analysis can then lead to prescriptions for overcoming the persistent training barrier.

A closer study of the development of proficiency during speed building requires a detailed understanding of how accuracy and response time change as a function of the rate of presentation (groups per minute); the serial position of a character within a group; the serial position of a character within a block; the identity of a character; the confusability of each character with all other characters; the times to perceive, recognize, and program a response to a stimulus; the sizes of the various memories involved in processing; and the "decay" times of the items in these memories. This detailed understanding can only come from a precise specification of the cognitive activity underlying Morse intercept behavior.

Qualitative Models. Toward this end, several models of processing in the Morse intercept task have been proposed (Sabol, Wisher and Medici, 1991; Townsend, 1991; Wisher, Kern and Sabol, 1990). These models all derive from the two descriptions of processing first presented by Wisher, Kern and Sabol (1990). Briefly, in their early state cognitive process model, it was assumed that subjects could execute only one process at a time and that the buffers in front of the processors were limited to one item at most. In the advanced state model, Wisher et al. assumed that subjects could execute several processes simultaneously. In addition, they assumed that the buffer sizes were greater than one. We want to pay particular attention to this advanced state model.

In the advanced state model four processes were arranged in series: auditory perception, character recognition, motor organization, and response execution. In front of each processor was a buffer that held information as it was passed along through the system. Each buffer had a limit on the number of items that it could hold and items in each buffer could decay over time. Here decay refers to the natural loss of information if it is not rehearsed.

Note that the processing time for each server, decay times for items in each buffer and capacity limits of each buffer were left unspecified.

At the outset of processing, the stimulus (a sequence of square waves of sound energy or, more simply, a tone sequence) was placed in the auditory buffer (echoic memory). In the auditory perception stage (which accepted information from the auditory buffer), it was assumed that the sound energy was transformed into a string of elements (i.e., that sound energy was transformed into a string of dits and dahs). These element strings were then passed to the recognition buffer. In the recognition stage (which accepted items from the recognition buffer), it was assumed that an element string was recognized as a character. The character code was passed on to the motor organization buffer. It was then assumed that the motor program needed to type the particular character was set up by the motor organization stage. Finally, the motor program was passed on to the motor execution buffer where, eventually, the response was initiated. We should note that the above reflects the details of just one of the late stage models proposed by Wisher et al. Several variations on the above were suggested.

Analytic Models. The above qualitative model suggests a number of points in the processing of stimuli where the operator may have difficulty. However, it is difficult to test the various hypotheses without being able to predict how accuracy and response time change as a function of the changes in the independent variables mentioned above. Towards this end it is necessary to quantify the model. Unfortunately, this is not by any means an easy task.

Briefly, the model proposed by Wisher et al. (1990) is quite clearly a queueing model. The literature on queueing models is extensive (e.g., Gross and Harris, 1985, discuss the basic elements of queueing theory; Rouse, 1980, reviews queueing networks that have been used to model person-machine systems). However, the majority of the existing analyses assume that the queueing system has reached steady state or equilibrium. Such will not be the case for the behavior of the queueing system in the tasks undertaken by the Morse intercept operator. In these tasks, the system is unlikely to reach equilibrium since five tone sequences are presented one at a time, followed by an interval of time which in many cases is long enough for the system to clear

itself. And even were the system not to clear itself, the fact that a break exists between groups makes it impossible to accept the assumption of steady state. Thus, one needs to undertake a transient analysis of the queueing system.

Townsend (1990), building on results reported in Fisher and Smith (1987), shows the form that the analyses must take. Basically, it is necessary to translate the queueing network into an Order-of-Processing (OP) diagram (Fisher and Goldstein, 1983; Goldstein and Fisher, 1991). Once so translated, it is possible to obtain closed form expressions for relatively restricted queueing networks. The analysis for two networks was worked out in detail by Townsend.

Briefly, in the first network, there were two nodes. A node consists of a server and a buffer (possibly of size zero). Townsend assumed that the server at the first node encodes and recognizes the stimulus and the server at the second node organizes and executes the response. The buffer at the first node was unlimited in capacity. The buffer at the second node was of size zero. If an item was encoded and the response server was executing, that item was held at the encoder and the encoder was prohibited from accepting further input until the response server completed its execution of the downstream item. Townsend assumed that n items were present in the first buffer at the start and that no new items were added to the system. Finally, Townsend assumed that the service times were independent, exponentially distributed random variables. For this system, he derived analytic expressions for the time on average it took to respond to the i^{th} item in the first buffer.

The above system does not contain a mechanism for producing errors. Thus, by itself, the system cannot be the one that subjects are using since, among other things, subjects always produce a significant number of errors. One reasonable way to generate errors is to assume that the subjects can execute at most one process in the system at any one time. If an item arrives while either the encoder or responder is busy, that item gets lost from the system. In the previous model, the item was simply held at the encoder. For this new system, Townsend was able to derive both the probability of an error

and the time on average it takes to respond, given that a correct response is made (he assumed that if an item was lost from the system either no response was made or an incorrect response was made).

Goal and Objectives

Goal. There are many reasons operators may find it difficult to complete training. One hypothesis (Wisher et al., 1990) is that some operators cannot simultaneously listen to (encode) an incoming signal at the same time as they attempt to recognize and respond to other signals that have already arrived. This ability is referred to as the ability to copy behind. The overall goal of the work undertaken as part of this effort is to determine the extent to which the copy behind hypothesis is supported by the data which has been collected in the Morse code copy task.

Objectives. Attainment of the above goal requires the meeting of three objectives. First, in order adequately to test the copy-behind hypothesis, it is necessary to model Morse code copy behavior. In particular, it is necessary to have a detailed model of the processes, memories, and other cognitive building blocks involved in the Morse intercept process. As noted above, Wisher et al. (1990) has proposed a rather extensive qualitative model. What is needed at this point is a more precise characterization of what exactly is meant by copy-behind, when exactly a period response will be produced, when no response will be made, and so on (Objective 1).

Second, once the generic models are available, it is necessary to quantify the behavior of the models. In particular, we were interested in having the model predict the probability that an operator types a character correctly, types a character incorrectly, types a character as a period or fails to type a character. (Periods are typed when an operator realizes that a character has been transmitted, but is unable to respond quickly enough. The period serves to maintain the format, i.e., the spacing.) And we were interested in having the model predict the time that it takes an operator to type a character correctly, the time that it takes an operator to type a character incorrectly, and the time that it takes an operator to type a period. As above, Townsend (1990) has produced analytic expressions for response times

and errors for several more simple models than the one developed by Wisher. What is needed at this point are predictions of the dependent variables for the more complex model. This can be achieved using either simulation (Objective 2a) or, where possible, deriving the relevant analytic expressions (Objective 2b).

Finally, given that the behavior of the models can be quantified, it is necessary to determine just how well the models fit the data. None of the models, either the qualitative model proposed by Wisher et al. (1990) or the quantitative models described by Townsend (1990) has been fit to results. What is needed now is an evaluation of just how well the various quantitative models do indeed fit the data (Objective 3).

A MODEL OF MORSE CODE COPY

The models of Morse intercept that we developed correspond closely to the advanced state models proposed by Wisher et al. (1990). To begin, we want to talk about the most general model (GenCOPB) that we developed, one which allows for copy-behind and large-capacity buffers at each stage. We then want to describe the two models that we tested. Both assume no buffering of the input at any stage. The first allows for copy-behind (LimCOPB); the second does not allow for copy-behind (NoCOPB).

There are a total of 77 parameters in the GenCOPB model, 14 of which are free and 63 of which are fixed by the data and/or condition. There are a total of 68 parameters in both the LimCOPB and NoCOPB models, 5 of which are free and, again, 63 of which are fixed by the data and/or condition. These parameters are described in detail in Appendix A. They are also described in context in the material below.

We assume throughout that the various server and decay times are independent and that their distribution is well described as a gamma. Briefly, the density function of a gamma is given by the right hand side of the equation below:

$$f(x) = [\beta^{-\alpha} x^{\alpha-1} e^{-x/\beta}]/\Gamma(\alpha) \qquad \text{if } x > 0,$$

$= 0$ otherwise.

The parameter α is referred to as the shape parameter; β is referred to as the scale parameter. The mean and variance are given by, respectively:

$$E[X] = \alpha\beta,$$

$$VAR[X] = \alpha\beta^2.$$

The function, $\Gamma(\alpha)$, is equal to $\alpha!$ when α is a nonnegative integer.

General Copy Behind Model (GenCOPB)

The Stages. We begin by developing a very general model, the GenCOPB, which allows copy behind. Specifically, we assume the existence of three stages, perception, recognition and execution. Each stage consists of a "server" (some process) and a "buffer" (some memory). Specifically, we assume the existence of three servers. We assume the existence of three buffers, one in front of each of the servers. And we assume that decay can occur in any one of the three buffers.

We assume that the stages are in series, where the string of tones input to the perception stage is output as a string of elements (dits and dahs). This string of elements is input to the recognition stage and output as a character code. And finally, the character code is input to the execution stage and output as an actual response.

It will be useful to have a more formal representation for the various inputs. To begin, note that there are 31 different possible tone inputs, 26 letters and 5 special characters (the 10 digits were not included when fitting the model and thus are not discussed). Each tone input t_i is associated with a particular element string e_i. Each element string e_i is associated with a particular character code c_i. And each character code c_i is associated with a particular response r_i. For example, the tone consisting of three 50 ms pulses is associated with the element string, di-di-dit. The element string di-di-dit is associated with the code for the character s. And the code for the character s is associated with the motor program for pressing the letter s on the keyboard.

Given this notation, a signal is output correctly if tone t_i is the input and response r_i is the output; a signal is output as an incorrect character if tone t_i is input and response r_j is the output ($j = 1,\ldots,31,\ i \neq j$). A signal is output as a period if tone t_i is the input and a period is the output (a period is indicated by the integer 44). And a signal is output as no response if tone t_i is the input and no response is made to this tone.

The Servers. Implicit in the above discussion of the stages is the following description of the role played by each of the servers. Specifically, as in the model proposed by Wisher et al. (1990), the perception server maps sound energy into a string of Morse elements, i.e., dits and dahs. The recognition server maps the string of Morse elements into a character. And the execution server both organizes a response and executes it.

We made several simplifying assumptions about the distributions of the service times. To begin, we assumed that the duration T_r of the recognition process and the duration T_e of the execution process did not depend on the identity of the input to the process or on the output of the process. Furthermore, given our work with other, similar cognitive tasks, it seemed reasonable to assume that the durations of the recognition and execution processes were independent, gamma distributed random variables. We let α_r and β_r represent the shape and scale parameters for the distribution of the duration of the recognition process; and we let α_e and β_e represent the shape and scale parameters for the distribution of the duration of the execution process. These four parameters were free.

Although we did not assume that the durations of the recognition and execution processes depended on the input to these processes, we did assume that the duration $T_p(e_j|t_i)$ of the perception process was dependent on the identity of the tone sequence t_i which was given as input. At this point we do not assume that the perception time depends on the output of the process. However, we believe that such a dependency will be necessary to incorporate in future

models and thus we keep this possibility open here by writing $T_p(e_j|t_i)$ instead of writing more simply, $T_p(t_i)$. Again, we assumed that the durations of the perception process for each tone sequence were independent, gamma distributed random variables. We used 32 parameters to describe these distributions, 31 shape parameters $\alpha_p(i)$, $i = 1,\ldots,31$ and one common scale parameter β_p. These parameters were fixed by the data in the latency confusion matrix (see Appendix A for a discussion of the way the parameters were set; see Appendix B for a discussion of the latency confusion matrix).

The Buffers. We made several critical assumptions about the buffers. Specifically, we assumed that when a stimulus arrived at the perception buffer and the perception buffer was empty, it went immediately to the perception server. We assumed that when a stimulus arrived at the perception buffer and the buffer had space available for it, the stimulus queued for service on a first come, first serve basis. And we assumed that when a stimulus arrived at the perception buffer and the buffer was full, the stimulus was lost from the system. This generated no response from the model. Similar remarks apply to stimuli arriving at the recognition and execution buffers.

We also assumed that when a stimulus decayed from any buffer, the stimulus was lost, regardless of the level of processing which had been completed immediately prior to the loss. This loss also generates a no response in the model.

A total of six parameters were needed to describe the decay times at each of the three buffers. Specifically, we assumed that the decay times were independent, gamma distributed random variables. The shape and scale parameters of the distribution of the decay time in the perception buffer are noted by, respectively, α_p^* and β_p^*. The shape and scale parameters of the distribution of the decay time in the recognition buffer are noted by, respectively, α_r^* and β_r^*. And the shape and scale parameters of the distribution of the decay time in the execution buffer are noted by, respectively, α_e^* and β_e^*. These six parameters are free.

A total of three parameters were needed to describe the capacity size of the three buffers. Specifically, the capacity sizes of the perception, recognition and execution buffers are noted by, respectively, γ_p, γ_r and γ_e. These three parameters are free.

Critical Perception Time. We made one critical assumption about the perception server, an assumption which produced a period response from the model. Specifically, we assumed that if the time to perceive the stimulus was longer than some critical value, say δ, then the model coded the incoming stimulus as a period. The rationale for doing so is the following. As the time it takes an operator to perceive a stimulus increases, the data suggest that accuracy decreases. Thus, rather than make an incorrect response, it may be optimal for the operator to produce a period as a response (which, during training, earns the operator a smaller point loss than an incorrect response).

Note that we assume that the duration of the perception process continues beyond the critical value δ. It is only after the perception process has completed that a comparison is made between the duration of this process and the critical value. This may appear counterintuitive at first glance. However, note that it could well be less time consuming simply to perceive a stimulus and then determine whether the critical value has been exceeded than concurrently both to perceive the incoming stimulus and to monitor the duration of this process. The exhaustive scanning mechanism proposed by Sternberg (1969) requires for its justification a similar argument. The critical time was a free parameter.

Character Errors. Define a character error as a transformation of a character (as opposed to period) input at one stage to a character output at the same stage which is not associated with the input. So, a character error at the perception stage is a transformation of a tone sequence t_i into an element sequence e_j, where $i,j = 1,\ldots,31$, $i \neq j$. A character error at the recognition stage is a transformation of an element string e_i into a character code c_j, where $i,j = 1,\ldots,31$, $i \neq j$. And a character error at the execution

stage is a transformation of a character code c_i into a response r_j, where $i,j = 1,\ldots,31$, $i \neq j$.

We need to consider the possibility of character errors at all three stages. To begin, consider character errors at the perception stage. When the duration of the perception server was less than δ, on some trials the perception stage worked correctly (i.e., tone sequence t_i was mapped into element string e_i); on other trials it produced a character error. Since there were 31 tone sequences, we need to specify for the perception stage the 31 probabilities $p(e_i|t_i)$, $i = 1,\ldots,31$. These 31 probabilities were determined by the error confusion matrix (see Appendix A for a discussion of how the parameters were estimated; see Appendix B for a discussion of the error confusion matrix). To keep the analysis relatively simple, we grouped the off-diagonal confusions. Thus, the probability of an incorrect perception transformation was set equal to $1 - p(e_i|t_i)$.

Next, we need to consider the possibility of character errors at the recognition and execution stages. Implicit in the above is the assumption that all confusions are bundled into the perception stage. That is, we do not also allow confusions at the recognition or execution stages. This is not because we believe that such confusions do not occur. Rather, it is because the preliminary data suggest that the majority of confusions occur at the perception stage. Of course, if our simplifying assumption is wrong, we should find the GenCOPB unable to account for significant aspects of the data.

Limited Copy Behind (LimCOPB) and No Copy Behind (NoCOPB) Models

The limited copy behind (LimCOPB) is a special case of the general copy behind (GenCOPB) model. Specifically, in the LimCOPB model, we assume that the buffers are of size zero at each of the perception, recognition, and execution stages. Note that this means we do not need parameters to describe the duration of the decay in these three buffers.

Strictly speaking, the no copy behind (NOCOPB) model is not a special case of the GenCOPB model since in the NoCOPB model all downstream processors are surveyed, not just the processor at the next stage. Specifically, if there are no items on any one of the perception, recognition or execution servers, then an arriving item is placed on the perception server. Otherwise, the arriving item is lost from the system and not recoverable.

SIMULATION

We now want to describe how we simulated the behavior of the three models, the GenCOPB, LimCOPB, and NoCOPB. Recall that we are interested in seven dependent variables, four related to accuracy and three related to time. Specifically, we are interested in simulating the probability of a correct response, the probability of an incorrect character response, the probability of an incorrect period response, and the probability of no response. In addition, we are interested in simulating the time that it takes to make a correct response, the time that it takes to make a incorrect character response, and the time that it takes to make an incorrect period response.

Because the general copy-behind model was complex, we decided first to simulate its behavior. The simulation follows directly from an OP representation of processing in such a task (Fisher and Goldstein, 1983; Goldstein and Fisher, 1991). To begin, we describe the state variables used in the simulation. We then describe the actual transition rules. Finally, we describe where in the simulation we obtain the relevant dependent variables. (Note that we will wait until the discussion of the analytical work to describe the actual OP diagram.) The source code is listed in Appendix C.

States

Conceptually, it is a straightforward matter to simulate the behavior of the system we have described. This is because for purposes of the simulation we can classify the evolution of the system over time into a finite number of states. Furthermore, the relevant state variables themselves are constant throughout the duration of any given state. Thus, we can simulate the system as a series of discrete events.

Each state s_i is defined as a structure consisting of a $(\gamma_a + 1)$ X 5 arrival matrix $M_a(i)$, a $(\gamma_p + 1)$ X 5 perception matrix $M_p(i)$, a $(\gamma_r + 1)$ X 5 recognition matrix $M_r(i)$, and a $(\gamma_e + 1)$ X 5 execution matrix $M_e(i)$. The first row of the arrival (perception, recognition, execution) matrix contains information on the item currently on the arrival (perception, recognition, execution) server. The second and subsequent rows contain information on the items in the arrival (perception, recognition, execution) buffer.

As noted above, the 31 characters are represented by the integers 1 - 31 (and the digits by the numbers 32 - 41). The intercharacter interval is represented by the number 42. The intergroup interval is represented by the number 43. The period character is represented by the number 44. Note that we will refer to the generic entity which passes through the system as an item, i.e., a tone sequence, element string, character code and response are all items.

Arrival. Consider just the arrival matrix for state s_i. It may help some to display a particular matrix and refer to the entries in the matrix as we discuss them. To keep things simple, we will display only two rows in the arrival matrix:

$M_a(i)$:

23	23	1	49	0	(server entries)
17	17	2	∞	0	(buffer entries)

To begin, consider the entries in the first row. The number (23) in the first column and first row indicates the identity of the tone sequence in state s_i which is currently on the arrival server. The number (23) in the second column and first row indicates the identity of the tone sequence which originally was sent. In the arrival matrix, this entry and the preceding entry are identical since they reference the same item before any processing has begun. The entry (1) in the third column and first row indicates the serial position of the arriving tone sequence in the group in which it was sent. Since there are five tone sequences in a group, the serial position

will always lie between 1 and 5, inclusive. The number (49) in the fourth column and first row indicates the amount of time which remains before the tone sequence identified in columns 1 and 2 actually arrives. The number (0) in the fifth column and first row indicates the amount of time the tone sequence will have spent in the system when it finishes processing in the arrival stage. This time is zero in the arrival matrix because it is assumed that timing does not begin until the arrival process is complete.

Consider next the second and subsequent rows in the arrival matrix. The entries (17 and 17) in the first and second columns of row 2 are identical and indicate, respectively, the identity of the next tone sequence which will be sent to an operator and the identity of the tone sequence which was originally sent to the operator. Note that the next tone sequence sent to the operator is the first tone sequence in the arrival buffer. The number (2) in the third column and second row indicates the serial position in the group in which it was sent of the tone sequence which is at the top of the arrival buffer. If the serial position of the arriving tone sequence lies between 1 and 4, then the serial position of the first tone sequence in the buffer is increased by 1. If the serial position of the arriving tone sequence is at the end of a group (i.e., is 5), then the serial position of the first tone sequence in the buffer will be 1. The serial position of an intercharacter or intergroup blank is set at 0. The entry (∞) in the fourth column and second row indicates the time remaining before the tone sequence which is at the top of the arrival buffer decays. We assume that this time is infinite, i.e., that there is no decay in the arrival buffer. Finally, the number (0) in the fifth column and second row indicates the total time that the tone sequence at the top of the arrival buffer will have spent in the system when the tone sequence currently being processed completes service. As above, this is zero because it is assumed that timing does not begin until the arrival is complete.

Perception. The entries in the perception, recognition and execution matrices are defined in much the same fashion as the above entries. To begin, consider the entries in the perception matrix and, in particular, the entries in the first row. Again, it may help to display an actual matrix, one where we assume that the buffer can hold at most one tone sequence:

$M_p(i)$:

16	16	2	123	650	(server entries)
8	8	4	500	150	(buffer entries)

The number (16) in the first column and first row indicates the identity of the tone sequence in state s_i which is currently being mapped to an element string. The number (16) in the second column and first row indicates the identity of the tone sequence which originally was sent. Since the arrival stage always outputs the same tone sequence which was sent as input, the tone sequences in the first and second columns are identical in the perception matrix (as well as the arrival matrix). The entry (2) in the third column and first row indicates the serial position of the tone sequence being perceived in the group in which it was sent. The number (123) in the fourth column and first row indicates the amount of time which remains before the tone sequence in the process of being perceived actually completes this process. The number (650) in the fifth column and first row indicates the amount of time the tone sequence being perceived will have spent in the system when it finishes processing in the perception stage. This time will be the sum of the times that it spent in the perception buffer and on the perception server. Thus, for example, if tone sequence 16 arrived at the perception stage when the server was occupied, if it took 200 ms to complete the servicing of the tone sequence currently on the server, and if it took 450 ms to service tone sequence 16, then the amount of time tone sequence 16 will have spent in the system when it finishes processing in the perception stage will be 650 ms.

Consider next the second row in the perception matrix. The entry (8) in the first column of row 2 indicates the identity of the next tone sequence which will be sent to the perception server (i.e., the identity of the first item in the perception buffer). The entry (8) in the second column of row 2 indicates the identity of the tone sequence in the first row in the perception buffer which originally was sent to the arrival stage. Again, this is the same as the entry in the first column and second row since we assume that no errors are introduced by the arrival stage. The number (4) in the third column and second row indicates the serial position in the group in which it was sent of the tone sequence which is at the top of the perception buffer.

Note that in this example this number is not simply one more than the corresponding number in the first row and third column. What this indicates is that the third tone sequence in the group must have entered the perception stage at a time when the buffer was full. Thus, it was lost from the system. The number (500) in the fourth column and second row indicates the time remaining before the tone sequence which is at the top of the perception buffer decays. Finally, the number (150) in the fifth column and second row indicates the total time that the tone sequence at the top of the perception buffer will have spent in the system when the tone sequence currently on the perception processor completes its service. In this case, tone sequence 8 could have arrived when there was 150 ms left to process of tone sequence 16 (of course, other scenarios could have led to the same time). Similar remarks would apply to subsequent rows in the perception matrix were we to assume a larger buffer.

Recognition. Next, consider the entries in the recognition matrix and, in particular, consider the entries in the first row of the recognition matrix. Assume that the buffer can hold only one element string:

$M_p(i)$:

10	12	5	321	1250	(server entries)
44	14	1	400	621	(buffer entries)

The number (10) in the first column and first row indicates the identity of the element string in state s_i which is currently being recognized. The number (12) in the second column and first row indicates the identity of the tone sequence which originally was sent and which has been transformed into the element string now on the recognition server. Since the perception stage can introduce errors, these two numbers need not be identical (and are not in this example). The entry (5) in the third column and first row indicates the serial position of the element string which is on the recognition server in the group in which it was sent. The number (321) in the fourth column and first row indicates the amount of time which remains before the element string on the recognition server completes processing on this server. The number

(1250) in the fifth column and first row indicates the amount of time the element string and associated tone sequence will have spent in the system when the element string on the recognition server finishes processing. This time will be the sum of the times that the item spent in the perception and recognition buffers plus the sum of the times it spent on the perception and recognition servers.

Consider next the second row in the recognition matrix. The entry (44) in the first column of row 2 indicates the identity of the next element string which will be sent to the recognition server (i.e., the identity of the first element string in the recognition buffer). In this particular example, the entry represents a period response, an indication that the critical time on the perception server was exceeded for the corresponding tone sequence and thus the perception server, rather than generating an element string associated with a character output, generated an element string for a period as an output. The entry (14) in the second column of row 2 indicates the identity of the tone sequence originally sent which is now the first element string in the recognition buffer. The number (1) in the third column and second row indicates the serial position in the group in which it was sent of the element string which is at the top of the recognition buffer. The number (400) in the fourth column and second row indicates the time remaining before the element string which is at the top of the recognition buffer decays. Finally, the number (621) in the fifth column and second row indicates the total time that the element string at the top of the recognition buffer (and associated tone sequence) will have spent in the system when the element string current on the recognition processor completes its service. Similar remarks apply to subsequent rows in the recognition matrix were the buffer to hold more than one element string.

Execution. Finally, consider the entries in the execution matrix and, in particular, consider the entries in the first row of the execution matrix. Again, it may be helpful to describe these entries in the context of a particular example. Assume that the execution buffer can hold only one character code:

$M_e(i)$:

20	10	3	101	1600	(server entries)
18	18	4	20	2210	(buffer entries)

The number (20) in the first column and first row indicates the identity of the character code in state s_i for which a response is currently being organized. The number (10) in the second column and first row indicates the identity of the tone sequence which originally was sent and which has been transformed into element string 20 at the perception stage and character code 20 at the recognition stage. The entry (3) in the third column and first row indicates the serial position of the character code which is on the execution server in the group in which it was sent. The number (101) in the fourth column and first row indicates the amount of time which remains before the character code on the execution server completes processing on this server. The number (1600) in the fifth column and first row indicates the amount of time the character code and its associated element string and tone sequence will have spent in the system when character code finishes processing in the execution stage. This time will be the sum of the times that the item spent in the perception, recognition and execution buffers plus the sum of the times it spent on the perception, recognition and execution servers.

Consider next the second row in the execution matrix. The entry (18) in the first column of row 2 indicates the identity of the next character code which will be sent to the execution server (i.e., the identity of the first character code in the execution buffer). The entry (18) in the second column of row 2 indicates the identity of the tone sequence which through processing in the perception and recognition stages has been transformed into the character code at the top of the execution buffer. The number (4) in the third column and second row indicates the serial position in the group in which it was sent of the character code which is at the top of the execution buffer. The number (20) in the fourth column and second row indicates the time remaining before the character code which is at the top of the execution buffer decays. Finally, the number (2210) in the fifth column and second row indicates the total time that the character code at the top of the execution buffer (and its associated element string and tone sequence) will have spent in the

system when the stimulus current on the execution processor completes its service.

Transition Rules

The simulation of the system requires knowledge of the state variables, as explained above. It also requires knowledge both of the events which lead to a transition between states and of the rules used to relate states which follow one another. We now want to describe these events and rules.

Briefly, a transition between states occurs when an item (i.e., a tone sequence, element string or character code) completes service or an item decays. The determination of which item will complete or decay first can be made in a straightforward fashion from the entries in the fourth column of the arrival, perception, recognition and execution matrices. Specifically, the minimum time is selected from the set of times consisting both of the arrival, perception, recognition and execution service times and of the perception, recognition and execution decay times.

Arrival. We now need to specify exactly what changes are made when anyone of the above events occur. Although quite tedious, the details vary enough for the different stages to require a relatively complete rendering of the transition rules. To begin, consider the arrival stage. The item in the arrival stage which completes first can be either a character or a blank. We need to consider both. And the item which completes first in the arrival stage can find the perception buffer full or not full. Again, we need to consider both possibilities.

i) Assume that the item which completes first is a character in the arrival stage. An example of a state, say s_i, where the first item to complete is a tone sequence in the arrival stage is given below on the left; the entries in the new state, say s_j, which follows it are given on the right. For the sake of simplicity, we assume that the buffers in the perception, recognition and execution stages can hold only one item. And we represent only the top most item in the arrival buffer:

Arrival Stage (s_i)

23	23	1	49	0
26	26	2	∞	0

Arrival Stage (s_j)

26*	26*	2*	250*	0
22*	22*	3*	∞	0

Perception Stage (s_i)

19	19	5	305	600
-	-	-	-	-

Perception Stage (s_j)

19	19	5	256*	600
23*	23*	1*	1200*	256*

Recognition Stage (s_i)

14	14	3	200	953
11	11	4	780	876

Recognition Stage (s_j)

14	14	3	151*	953
11	11	4	731*	876

Execution Stage (s_i)

8	8	1	510	1451
9	9	2	452	1008

Execution Stage (s_j)

8	8	1	461*	1451
9	9	2	403*	1008

As noted above, in order to determine which event completes first, we need to look at the entries in column four of the matrices defining state s_i and find the minimum. Doing such, we see that the times remaining on the arrival, perception, recognition and execution servers are respectively 49, 305, 200 and 510 ms. And we see that the times remaining before decay occurs in the arrival, recognition and execution buffers are, respectively, ∞, 780 and 452 ms (the perception buffer is empty, as indicated by the "-"). Thus, the minimum time to completion or decay is 49 ms, or the remaining arrival time of tone sequence 23.

The changes in the arrival, perception, recognition and execution matrices are given on the right in the example state s_j. The starred (*) entries indicate where changes occur. To begin, consider the change in the arrival matrix. Since tone sequence 23 completes, we move it off the arrival server. We move the first tone sequence in the arrival buffer, tone sequence 26, to the arrival server and assign it the appropriate arrival time (say 250). Note that we have displayed only the first two rows of the arrival

matrix. However, we are assuming that the arrival buffer contains additional items, and in particular, contains tone sequence 22 below tone sequence 26. Thus, we move tone sequence 22 up to the first position in the arrival buffer. By implication it is the third tone sequence in the current group.

We also need to change the perception matrix. Note that although the server of the perception matrix is occupied, the buffer of the perception matrix is empty. Thus, we can put the tone sequence which just arrived (23) in the buffer of the perception matrix and assign it a decay time, say 1200. We need to make two additional changes. First, the time remaining to process the tone sequence on the perception server must be reduced by 49 ms, from 305 to 256 ms. Second, the amount of time the newly arrived tone sequence 23 will have spent in the perception buffer after the tone sequence currently on the server completes its execution must be assigned. In this case, this is simply 256 ms. Note that the amount of time that tone sequence 19 will have spent in the system after it completes processing is unaffected by the completion of the tone sequence in the arrival stage. Thus, the entry in the perception matrix in column 5 and row 1 in both states s_i and s_j is 600.

Finally, we need to decrement the server and decay times in the recognition and execution matrices. For example, the time remaining until element string 11 decays in the recognition buffer is 780 ms when we first examine the system. After tone sequence 23 arrives, 49 ms will have elapsed. Thus the time remaining until element string 11 decays is reduced from 780 to 731 ms.

ii) Above, a character in the arrival stage was the first item to complete. It could have been a blank (an intercharacter or intergroup blank) instead. An example is given below (recall that 42 is the code for an intercharacter blank):

Arrival Stage (s_i)

42	42	0	49	0
26	26	1	∞	0

Arrival Stage (s_j)

26*	26*	1*	250*	0
22*	22*	2*	∞	0

Perception Stage (s_i)

19	19	5	305	600
-	-	-	-	-

Perception Stage (s_j)

19	19	5	256*	600
-	-	-	-	-

Recognition Stage (s_i)

14	14	3	200	953
11	11	4	780	876

Recognition Stage (s_j)

14	14	3	151*	953
11	11	4	731*	876

Execution Stage (s_i)

8	8	1	510	1451
9	9	2	452	1008

Execution Stage (s_j)

8	8	1	461*	1451
9	9	2	403*	1008

In this example, note that all the times are changed as above and that a new tone sequence is added to the arrival server. However, the blank is not moved to the perception buffer since blanks are not processed through the system.

iii) Finally, suppose that the item which completes first is on the arrival server and that the perception buffer is full. Then, we make the changes indicated below:

Arrival Stage (s_i)

23	23	1	49	0
26	26	2	∞	0

Arrival Stage (s_j)

26*	26*	2*	250*	0
22*	22*	3*	∞	0

Perception Stage (s_i)

19	19	4	305	600
16	16	5	838	403

Perception Stage (s_j)

19	19	4	256*	600
16	16	5	787*	403

Recognition Stage (s_i)

14	14	2	200	953
11	11	3	780	876

Recognition Stage (s_j)

14	14	2	151*	953
11	11	3	731*	876

Execution Stage (s_i)				
8	8	5	510	1451
9	9	1	452	1008

Execution Stage (s_j)				
8	8	5	461*	1451
9	9	1	403*	1008

Note that the only difference here is that tone sequence 23 is not added to the perception buffer since this buffer is already full.

Perception. Next, we need to consider the perception stage. An event (i.e., a completion or decay) in this stage will drive the transition if either the item on the perception server completes or an item in the perception buffer decays before any other service or decay occurs. To begin, assume that the event which leads to the transition is the completion of the processing of the tone sequence on the perception server. Then, if the duration of the service is greater than some critical time, we assume that the tone sequence on the server is replaced by a period response. For illustrative purposes, let us assume that this time is 1600 ms. If the duration of the perception process is less than 1600 ms, then the tone sequence t_i on the perception server is replaced with its correct element string, e_i, with probability $p(r_i|t_i)$ and with an incorrect element string e_j $(i \neq j)$ with probability $1 - p(r_i|t_i)$. Furthermore, it will be the case that sometimes the tone sequence on the perception server completes (regardless of its duration) when the recognition buffer is empty; at other times it will complete when the recognition buffer is full.

As noted above, it may be the decay of a tone sequence in the perception buffer (rather than the processing of a tone sequence on the perception server) that drives the next transition. If this is the case, then we need to enforce alternative transition rules.

The full set of rules is discussed below.

i) To begin, assume that it is the perception service time which is the minimum of the times remaining until an item either completes service or

decays from a buffer. And assume the perception service time is greater than the critical time of 1600 ms. Then, the matrices are changed as in the following example:

Arrival Stage (s_i)

23	23	1	350	0
26	26	2	∞	0

Perception Stage (s_i)

19	19	5	49	1888
-	-	-	-	-

Recognition Stage (s_i)

14	14	4	200	953
-	-	-	-	-

Execution Stage (s_i)

8	8	2	510	1451
9	9	3	452	1008

Arrival Stage (s_j)

23	23	1	301*	0
26	22	2	∞	0

Perception Stage (s_j)

-*	-*	-*	-*	-*
-	-	-	-	-

Recognition Stage (s_j)

14	14	4	151*	953
44*	19*	5*	731*	151*

Execution Stage (s_j)

8	8	2	461*	1451
9	9	3	403*	1008

There are several things to note. First, the tone sequence 19 on the perception server now becomes the element string 44 in the recognition buffer (the string associated with a period response). Second, the actual service time in the perception stage is greater than 1600 ms and, in this case, is equal to 1888 ms, assuming that no time was spent in the perception buffer.

ii) Assume that the first item to complete or decay is in the perception stage, as above. However, now assume that the perception service time is less than the critical perception time of 1600 ms. Then, as noted above, one of two things can happen. The tone sequence 19 can be transformed correctly into the element string 19 or it can be transformed into some other element string. When the element string was transformed incorrectly, we mapped tone sequence 1

to tone sequence 2 and tone sequences 2,...,31 to tone sequence 1. For example, below tone sequence 19 is mapped to tone sequence 1:

Arrival Stage (s_i)

23	23	1	350	0
26	26	2	∞	0

Perception Stage (s_i)

19	19	5	49	742
-	-	-	-	-

Recognition Stage (s_i)

14	14	4	200	953
-	-	-	-	-

Execution Stage (s_i)

8	8	2	510	1451
9	9	3	452	1008

Arrival Stage (s_j)

23	23	1	301*	0
26	22	2	∞	0

Perception Stage (s_j)

-*	-*	-*	-*	-*
-	-	-	-	-

Recognition Stage (s_j)

14	14	4	151*	953
1*	19*	5*	731*	151*

Execution Stage (s_j)

8	8	2	461*	1451
9	9	3	403*	1008

Note that the time tone sequence 19 will have been in the system when it completes service on the perception server is only 742 ms, as opposed to 1888 ms in the previous example. Thus, a period response is not generated.

iii) In both of the above examples, there was room in the recognition buffer for the output of the perception server. This will not always be the case. When this happens, the times are updated appropriately and the element string output from the perception server is bumped from the system, incrementing by one the number of no responses in the simulation. We have already described how this change is made when we discussed the transition rules for the arrival stage.

iv) Finally, suppose that it is the decay of the tone sequence in the perception buffer that drives the transition. Then, the state entries might appears as below:

Arrival Stage (s_i)

23	23	1	386	0
26	26	2	∞	0

Perception Stage (s_i)

19	19	4	305	600
16	16	5	49	403

Recognition Stage (s_i)

14	14	2	200	953
11	11	3	780	876

Execution Stage (s_i)

8	8	5	510	1451
9	9	1	452	1008

Arrival Stage (s_j)

23	23	1	337*	0
26	26	2	∞	0

Perception Stage (s_j)

19	19	4	256*	600
-*	-*	-*	-*	-*

Recognition Stage (s_j)

14	14	2	151*	953
11	11	3	731*	876

Execution Stage (s_j)

8	8	5	461*	1451
9	9	1	403*	1008

Note that tone sequence 16 in the perception buffer has decayed and thus appears in state s_i, but not in state s_j.

Recognition and Execution. The transition rules in the recognition and execution stage are similar to those in the perception stage with one exception. In particular, the transformation of an element string into its corresponding character code is considered errorless and the transformation of each character code into its corresponding motor program is considered errorless.

Dependent Variables

Information on six of the seven dependent variables we obtain from the execution matrix at each point in time when the first item in a state to complete is on the execution server. An example execution matrix is listed below:

Execution Stage

8	8	1	510	1451	(server entries)
44	9	2	452	1008	(buffer 1 entries)
1	17	3	856	1217	(buffer 2 entries)

Now, consider how we obtain six of the seven measures from the above matrix. Note that 1451 is the final measure of the response time for tone sequence 8. This will be a sample of the correct response time (since the entries in columns 1 and 2 of row 1 are identical). Thus we increment by one the number of correct responses, giving us the information we need eventually to compute the probability of a correct response. When we return at some point in the future to this matrix, as long as the first character code in the buffer has not decayed, a period (44) will be output. In the fifth column of the first row will now appear the time it took to make the period response when tone sequence 9 was the original item. Thus, we have a sample of a period response time. Furthermore, we increment by one the number of period responses, giving us the information we need eventually to compute the probability of a period response. Finally, when we return at a still later point to this matrix, as long as the second character code in the buffer has not decayed, a 1 will be output. In the fifth column of the first row will appear the incorrect response time. And we increment by one the number of incorrect response, giving us the information we need eventually to compute the probability of an incorrect response.

We obtain information on the seventh dependent variable as the system evolves. That is, rather than keeping track of decayed items or items which have been bumped from the system because they were output at a time when the immediate downstream buffer was full, we simply increment by one the number of

items to which an operator will make no response at the time an item disappears. Thus, we obtain the information we need in order to compute the probability of no response.

ANALYSIS

Although the simulation as described above yields the desired predictions, it has one weakness (a weakness common to all simulations). Specifically, it is never truly possible to identify the parameter settings which maximize the fit of a given model to the data. The best that can be done with a simulation is to make the search through the parameter space a relatively fine one, hoping that there exist no unexplored points which would radically alter the fit of the particular model being simulated. Exact identification of the parameter settings which maximize the fit can only be done when closed form expressions can be obtained for the dependent variables of interest.

Unfortunately, the GenCOPB, LimCOPB and the NoCOPB are themselves extremely complex models, making it difficult if not impossible to obtain the desired closed form expressions. And even were one to obtain the expressions, they themselves might be too unwieldy to work with. The question at this point is therefore a threefold one. First, in principle can closed form expressions be obtained for the GenCOPB, LimCOPB and NoCOPB models? Second, if in principle such expressions can be obtained, are there some experimental paradigms which, unlike the current paradigm, introduce enough constraints to make it possible actually to derive these expressions in a reasonable amount of time? And third, how would one obtain these expressions were such paradigms to exist?

OP Diagrams

We can answer the first question quite easily. Indeed, it is possible to obtain closed form expressions for the various expressions in which we have an interest. In order to prove that this is the case, we first need to describe the general structure of an OP diagram. Then, we need to show that the state space of the models we have developed is itself an OP diagram. Once we know

the structure of our models is an OP diagram, we can use the work in Fisher and Goldstein (1983; Goldstein and Fisher, 1991) to obtain the desired expressions given certain auxiliary assumptions.

General Structure. Formally, an OP diagram is a structure $\langle \Gamma, \Psi, \Omega \rangle$ consisting of a set Γ of processes, set Ψ of paths, and set Ω of states. The set Γ consists of $n(\Gamma)$ processes, labelled $x_1, \ldots, x_{n(\Gamma)}$. Each state in Ω corresponds to a partition of the processes in Γ. The states are labelled $s_1, \ldots, s_{n(\Omega)}$. Note that not all partitions of Γ need be in Ω. Each path α in Ψ consists of a sequence of states. We identify the states on path α, in order, as $s_{\alpha_1}, s_{\alpha_2}, \ldots, s_{\alpha_{n(\alpha)}}$.

Several remarks need to be made about the states. First, the processes in each state s_i must be resident in one and only one of four sets, the pre-active set $A(s_i)$, the current set $C(s_i)$, the completed set $K(s_i)$, or the other set $O(s_i)$. Second, the first state along any path must have empty completed and other sets and at least one process in the current set. Third, the last state along any path must have empty pre-active and current sets and at least one process in the completed set.

There are restrictions not only on the character of the sets of states at the beginning and end of each path, but also restrictions on the character of states adjacent to one another on a path. These restrictions are displayed graphically in Figure 1 below:

Figure 1

Pre-Active	Current	Completed	Other
$A(s_i) \dashrightarrow A(s_j)$	$A(s_i) \dashrightarrow C(s_j)$	$A(s_i) \not\dashrightarrow K(s_j)$	$A(s_i) \not\dashrightarrow O(s_j)$
$C(s_i) \not\dashrightarrow A(s_j)$	$C(s_i) \dashrightarrow C(s_j)$	$C(s_i) \dashrightarrow K(s_j)$	$C(s_i) \dashrightarrow O(s_j)$
$K(s_i) \not\dashrightarrow A(s_j)$	$K(s_i) \not\dashrightarrow C(s_j)$	$K(s_i) \dashrightarrow K(s_j)$	$K(s_i) \not\dashrightarrow O(s_j)$
$O(s_i) \not\dashrightarrow A(s_j)$	$O(s_i) \dashrightarrow C(s_j)$	$O(s_i) \not\dashrightarrow K(s_j)$	$O(s_i) \dashrightarrow O(s_j)$

Note that an arrow, --->, joining two different sets in adjacent states, say $A(s_i)$ ---> $A(s_j)$, means that later set (in this case, the pre-active set $A(s_j)$ can contain processes in the earlier set $A(s_i)$. The negated arrow, -/->, connected two different sets in adjacent states, say $C(s_i)$ -/-> $A(s_j)$, means that the later set, $A(s_j)$, does not contain any elements of the earlier set, $C(s_i)$.

In words, if state s_j is an immediate successor of state s_i for some path in Γ, then restrictions apply to the four sets which define state s_j. First, the pre-active set $A(s_j)$ of state s_j can contain no processes which are not contained in the pre-active set $A(s_i)$ of state s_i. Second, the current set $C(s_j)$ of state s_j can contain no processes in the completed set $K(s_i)$ of state s_i. Third, the completed set $K(s_j)$ of state s_j must contain every process in the completed set $K(s_i)$ of state s_i as well as at least one process from the current set $C(S_i)$ of state s_i. Furthermore, no processes from the pre-active set $A(s_i)$ or other set $O(s_i)$ of state s_i can appear in the completed set $K(s_j)$ of state s_j. Finally, the other set $O(s_j)$ of state s_j can contain no processes in the pre-active set $A(s_i)$ or completed set $K(s_i)$ of state s_i.

If a network is an OP diagram, then using the techniques of Fisher and Goldstein (1983; Goldstein and Fisher), we can obtain expressions for the behavior of the network if two additional conditions are met. First, the number of states along any path in the OP diagram must be finite. And second, any process which becomes current in one or more states along a path must eventually enter a completed set.

We can easily show that the various models we have described can be represented as OP diagrams. Briefly, in our particular case, we associate seven processes with each tone sequence, four service processes (one each for the arrival, perception, recognition and execution servers) and three decay processes (one each for the perception, recognition and execution servers). At the outset, the pre-active set consists of each of the seven processes for

all of the different tone sequences except for the arrival service process for the first tone sequence. This will appear in the current set of the start state. When the arrival service process completes, this process enters the completed set of the second state and both the perception service process associated with the first tone sequence and the arrival service process associated with the second tone sequence enter the current set. And so on.

Analysis

In theory, since the various models can be represented as OP diagrams we can derive the moments of the response time analytically. However, this does not mean we can obtain closed form expressions for these models, or even if we could obtain such expressions, that we could do so in a reasonable length of time. In fact, although we cannot obtain closed form expressions as the models are currently set forth, we can easily modify the model so that such expressions can be obtained in theory. The problems requiring modification come at two points.

First, recall that in the perception stage we took one path if the perception service time were greater than the critical perception time, another path if this were not the case. Thus, knowledge of the last finishing process and the sequence of states up until the current state does not alone determine the path taken out of the current state. Strictly speaking then, the various models we have proposed cannot be represented as OP diagrams (Goldstein and Fisher, 1991), a seeming contradiction to our conclusion in the previous section.

Fortunately, we can easily get around the above problem. Specifically, we can represent the critical time itself as a process. The critical time process is then in a race with the perception process. If the critical time process finishes before the perception process, then the tone sequence on the perception server is replaced with a period when the server completes. If the perception process finishes before the critical time process, then the tone sequence on the perception server is replaced with its associated element string when the server completes.

Second, it is not possible to obtain closed form expressions for gamma distributed random variables unless the gamma is a sum of independent exponentials. This does not overly restrict the shape of the distributions and seems a reasonable enough assumption. Thus, below we assume that the distributions of the various processes are independent, gamma distributed random variables subject to the constraint that the gamma random variables are themselves sums of independent exponentials.

Feasibility. As noted above, it does not follow that because we can derive closed form expressions in principle, we will be able to do such in practice. In fact, it is much too time consuming to derive such expressions. This can be demonstrated easily enough. Briefly, if 250 stimuli (tone sequences) are displayed on any one trial, there are 250! different orders in which any one combination of tone sequences could be presented. Since the analytic expressions for the response time depend on both the order in which the tone sequences are presented and their identity, it simply is not possible to derive all 250! different predictions in any reasonable time frame.

Fortunately, there is a way around the above problem. Specifically, we need not predict response times for all 250 stimuli. We could predict response times for just the first five stimuli in a group. Since there are 31 characters, there are 31 X 30 X 29 X 28 X 27 = 20,389,320 different predictions of response time we would need to make (this assumes that no character appears twice in the same group of five). Again, this is obviously way to large. However, suppose that we were to use only five characters throughout an entire experiment. Each trial would consist of a random permutation of these five characters. A large gap would appear between a group, say 3 seconds, enough to clear the system. In this case we would need to make only 5! = 120 predictions of response time. This may be small enough to be manageable.

Note that we do not need a different closed form expression for each of the 120 different predictions of response time. The closed form expression remains the same across predictions. What changes are the values of the parameters we substitute into the closed form expression.

Computations. We need now to represent processing as an OP diagram in order to show how we would compute the desired closed form expression for response time. Consider just the NOCOPB model (it is simplest). And consider just one path in the evolution of the system. Suppose that five stimuli are presented, $t_1,\ldots,t_5$. Label the arrival, perception, critical time, recognition and execution processes associated with the i^{th} tone sequence respectively as a_i, p_i, c_i, r_i, and e_i. Then, an example path through the OP diagram would appear as below:

State	Current	Completed	Other
1	a_1		
2	a_2,p_1,c_1	a_1	
3	a_3,p_1,c_1	a_2	
4	a_4,p_1,c_1	a_3	
5	a_5,p_1,c_1	a_4	
6	a_5,r_1	p_1	c_1
7	a_5,e_1	r_1	
8	a_5	e_1	
9	p_5,c_5	a_5	
10	p_5	c_5	
11	r_{44}	p_5	
12	e_{44}	r_{44}	
13	---	e_{44}	

Note that we have not represented the pre-active set simply because to do so would be too cumbersome.

Briefly, the figure should be interpreted as follows. In state s_1 the arrival server is busy processing the first tone sequence t_1 on the arrival

server. In state s_2 this arrival completes (so a_1 is placed in the completed set) and the first tone sequence is passed on to the perception server (p_1). A process (c_1) is also placed on the perception server and represents the critical time. And finally, the next arrival (a_2) is now placed on the arrival server. Processing continues in this fashion for all five arrivals.

Note that in state s_5 we assume that the perception process (p_1) completes before the process (c_1) which represents the critical time. Thus c_1 is placed in the other set since its processing is interrupted. Just the opposite happens in state s_9. Here the critical perception time associated with tone sequence t_5 is shorter than the duration of the perception time for the same tone sequence. Thus, the fifth tone sequence is sent as a period element string (44) to the recognition server.

It is now a straightforward matter to derive the seven dependent variables. This requires knowing how to compute the probability that a particular path is taken and the expected duration of the path. The computations are reasonably simple if we assume that the durations of the various processes are independent, exponentially distributed random variables. In this case, we need only one parameter to describe each process x_i, the rate parameter, say λ_i.

The probability of taking a path from the start state up to and including state s_{α_i} is simply the product of i conditional probabilities, the first of which, $P(s_{\alpha_2} | s_{\alpha_1})$, is the ratio of the rate parameter of the process which completes when a transition is made from state s_{α_1} to state s_{α_2} to the sum of the rate parameters of the processes current in state s_{α_1}, the second of which, $P(s_{\alpha_3} | s_{\alpha_2})$, is the ratio of the rate parameter of the process which

completes when a transition is made from state s_{α_2} to state s_{α_3} to the sum of the rate parameters of the processes current in state s_{α_2}, and so on. The time on average it takes to traverse a path up to and including state s_{α_i} is a sum of i conditional expectations, the first of which, $E[T_{\alpha_1}|\alpha]$, is the reciprocal of the sum of the rate parameters in state s_{α_1}, the second of which, $E[T_{\alpha_2}|\alpha]$, is the reciprocal of the sum of the rate parameters of the processes current in state s_{α_2}, and so on.

Given the above information, we can go on to compute the moments of the response time (Fisher and Goldstein, 1983; Goldstein and Fisher, 1991). We will not develop these computations here since to do so would require a fair amount of work, work which may well change as we become more familiar with the sort of model we eventually want to fit.

MODEL EVALUATION

We were most interested in whether the LimCOPB or NoCOPB models better fit the results. In order to determine this, we performed some preliminary analyses. Below, we describe these preliminary analyses for the two models.

Briefly, we fit the results from 19 subjects. The subjects were trained at the US Army Intelligence School, Fort Devons, Massachusetts. The subjects had completed the character learning phase and were beginning the speed building phase at the starting rate of six groups per minute (5 characters appeared in each group). For these subjects, we had for each of the tone sequences t_i, i = 1 - 31, the probability that the subjects made a character response r_j, j = 1,...,31, a period response, or no response. And we had for each of the tone sequences t_i, i = 1 - 31, the time on average that it took the subjects to make a character response r_j, j = 1,...,31, or a period response. The relevant data are given in Appendix B.

LimCOPB

Parameter Search. There are five free parameters with the LimCOPB model. We did not engage in an extensive search of the parameter space since the fit proved quite good with our initial estimates. Specifically, we found a good fit with the following parameter settings for α_r, β_r, α_e, β_e and δ:

Process	Shape	Scale	Mean	Standard Deviation
Perception Server	{parameters determined from data; depended on individual tone sequence; see Appendix A}			
Perception Decay	{not applicable since there is no buffer}			
Recognition Server	15(α_r)	10(β_r)	150	39
Recognition Decay	{not applicable since there is no buffer}			
Execution Server	15(α_e)	10(β_e)	150	39
Execution Decay	{not applicable since there is no buffer}			

The critical time δ we set at 1500 ms after trying several alternatives. The other 63 parameters (32 parameters to describe the arrival times for each character; 31 parameters to describe the errors) were fixed by the confusion matrices in Appendix 2.

With the above parameter settings, we ran the simulation. The predicted and observed probabilities are displayed below. The simulated results are based on the output from 500 trials:

	P(CORRECT),	P(INC CHAR),	P(PERIOD),	P(NO RESP)
Predicted	0.848000	0.064000	0.068000	0.020000
Observed	0.849500	0.049745	0.069752	0.031003

Superficially, the agreement looks reasonable enough. We also obtained predictions of the relevant response times which are listed below together with the observations:

RT(CORRECT), RT(INC CHAR), RT(PERIOD), RT(NO RESP)

Predicted	1184.7144	1162.0707	1922.2770	0.0000
Observed	1230.8064	1528.5364	2080.2903	0.0000

Again, the match looks good except for the response times to the incorrect characters.

Incorrect Character Response Times. The fact that we did not fit the response times for the incorrect characters arises directly from the way in which we assign these response times. In particular, note that the time to process tone sequence t_i on the perception server is based on the time $T(r_i|t_i)$ that it takes to make a correct character response. If the perception process for tone sequence t_i takes longer than the critical perception time, 1500 ms in the above run, then it is assumed to be transformed into the element string corresponding to the period. Since only times greater than 1500 ms go into the average of the period response time, this response time will be at least as great as 1500 ms even though the underlying distribution from which the times were drawn may have a mean much smaller than 1500 ms. However, the predicted incorrect character response times will equal the correct character response times since they come from one and the same truncated distribution.

Specifically, if the perception process for tone sequence t_i takes less than 1500 ms, then the tone sequence is assumed to be transformed correctly with probability $\hat{p}(r_i|t_i)$ and incorrectly with one minus this probability. No adjustment is made to the incorrect response times however. Thus, we find a failure to predict the incorrect response times.

We need in future work to extend the LimCOPB model so that it can predict the incorrect response times. Two possibilities suggest themselves. First, we could introduce a second critical time with a value less than the first critical time. Thus, say we have δ_1 and δ_2, where $\delta_1 < \delta_2$. Then, a tone sequence would be transformed into a period element string if the perception service time were greater than δ_2. A tone sequence would be transformed into

an incorrect element string if the perception service time were greater than δ_1 and less than or equal to δ_2 . And a tone sequence would be transformed into the correct, associated element string if the perception service time were less than or equal to δ_1.

Alternatively, we could assign one distribution to the correct response times and a second distribution to the incorrect response times. Then, as above we would assume that any response time greater than the critical response time eventuated in a period response. Currently, we are exploring both ways of handling the incorrect character response times.

Serial Position Information. We should note before concluding this subsection that the simulation predicts not only the seven dependent variables, but also serial position information. Below, we see that there there appears to be a primacy effect, but not a recency effect, within groups of five:

SERIAL POSITION INFORMATION

47	43	40	42	40

This is what we would expect since the first item in a group is more likely to meet an unoccupied perception server than is the last item in a group.

We also generated predictions of response time across the five serial positions within a group:

SERIAL POSITION INFORMATION

1142.072	1130.202	1273.773	1170.171	1219.631

Since there is no room in any of the buffers, any item that made it through the system should have had a clear path all the way. Thus, we would not expect to see serial position effects. Without further statistical analysis it is difficult to determine whether the above differences are significant.

NoCOPB

Parameter Search. As might be expected, the NoCOPB model fit the results every bit as well as the LimCOPB model. This is expected since at the slower rates there is not as much need for copying behind. Below, we list the results of three runs, one with a critical time of 1600 ms, one with a critical time of 1400 ms, and one with a critical time of 1300 ms. The first run is with δ = 1600:

```
  CRITICAL TIME AT PERCEPTION SERVER    =    1600.000

            P(CORRECT), P(INC CHAR), P(PERIOD), P(NO RESP)
Predicted     0.892000    0.056000    0.004000    0.048000
Observed      0.849500    0.049745    0.069752    0.031003

            RT(CORREC), RT(INC CHR), RT(PERIOD), RT(NO RSP)
Predicted    1204.0682   1220.6215   2086.5015      0.0000
Observed     1230.8064   1528.5364   2080.2903      0.0000

  SERIAL POSITION INFORMATION: NUMBER CORRECT
         47          43          45          43          45

  SERIAL POSITION INFORMATION: CORRECT TIME
   1167.336       1214.751       1271.863       1187.347       1180.407
```

Note that the predicted response times (except for the incorrect characters) match well the observed response times. However, the observed probability of a period response (0.06975) is seriously underestimated (0.004).

If we want to increase this probability we need to decrease the critical perception time, which we do in the run below:

```
  CRITICAL TIME AT PERCEPTION SERVER    =    1400.000

            P(CORRECT), P(INC CHAR), P(PERIOD), P(NO RESP)
Predicted     0.832000    0.040000    0.068000    0.060000
Observed      0.849500    0.049745    0.069752    0.031003
```

	RT(CORREC),	RT(INC CHR),	RT(PERIOD),	RT(NO RSP)
Predicted	1152.5948	1265.1078	1830.5255	0.0000
Observed	1230.8064	1528.5364	2080.2903	0.0000

SERIAL POSITION INFORMATION: NUMBER CORRECT

43	43	42	42	38

SERIAL POSITION INFORMATION: CORRECT TIME

1154.037	1142.396	1125.010	1159.207	1185.682

The predicted probability (0.068) of a period response now matches closely the observed probability. But note that in raising the predicted probability of a period response we have lowered the average period response time to the point where it no longer is closely tied to the observed period response time. Thus, the model is clearly sensitive to variations in the parameters.

Finally, we would expect even worse fits were we to decrease the critical response time still further. In the run below, we lower this time to 1300 ms:

CRITICAL TIME AT PERCEPTION SERVER = 1300.000

	P(CORRECT),	P(INC CHAR),	P(PERIOD),	P(NO RESP)
Predicted	0.724000	0.044000	0.124000	0.108000
Observed	0.849500	0.049745	0.069752	0.031003

	RT(CORREC),	RT(INC CHR),	RT(PERIO),	RT(NO RSP)
Predicted	1169.4633	1225.6049	1854.8347	0.0000
Observed	1230.8064	1528.5364	2080.2903	0.0000

SERIAL POSITION INFORMATION: NUMBER CORRECT

38	31	37	37	38

SERIAL POSITION INFORMATION: CORRECT TIME

1110.812	1167.096	1199.870	1182.710	1187.541

Note that the probability of a correct response, the probability of a period response, and the probability of no response are now seriously off. It is clear why the probability of a period response should increase and why the probability of a correct response should decrease. It is not clear to us why the probability of no response should also increase. Specifically, the distribution of the perception service times does not change as a function of the critical perception time, nor do either the recognition or execution service times change. Thus, the probability of a period response ought to remain roughly the same. We need to determine whether the above variation is random or systematic, and if systematic why it is occurring.

CONCLUSION

The simulation developed here allows the user to predict for the Morse code copy task the probability of a correct response, the probability of an incorrect response, the probability of a period response, the probability of no response, and the probability of a correct response for each of the five serial positions in a group. The simulation also allows the user to predict the time that it takes to execute a correct response, the time that it takes to execute an incorrect response, the time that it takes to execute a period response, and the time that it takes to execute a correct response for characters occurring in each of the five serial positions of a group.

In the most general case the simulation required the estimation of 77 parameters, 63 of which were fixed by the data. These parameters were needed to describe the distribution of the duration of the service and decay times at each of the perception, recognition and execution stages, the number of stimuli which could fit into the three buffers, and the critical perception time.

No models to date have allowed for as wide a specification of input parameters or for as rich a set of predictions. These models should make it possible to explore in detail a number of hypotheses about the origins of the training problems with learning to copy Morse code at higher input rates.

One such hypothesis was tested, in particular, the hypothesis that subjects early in practice could not process two stimuli at the same time (the copy behind hypothesis). Several variants of the most general model were created to test this hypothesis. One, the LimCOPB, allowed copy behind; the other, the NoCOPB, did not. The models fit about equally well. This is not all that surprising given that copy behind is not particularly useful early on since the intercharacter interval is relatively large. It still remains to be determined whether at the faster rates subjects who make it through the training learn to copy behind whereas those that don't fail to learn this skill.

REFERENCES

Bryan, W.L. & Harter, N. (1897). Studies in the psychology and physiology of the telegraphic language. Psychological Review, 4, 26-53

DOES. (1990). Ability based Morse training. Directorate of Evaluation and Standardization Report No. 011, 4 Dec 1990, U.S. Army Intelligence School, Devens, Ft. Devens, Mass.

Fisher, D. L. and Goldstein, W. M. (1983). Stochastic PERT networks as models of cognition: Derivation of the mean, variance and distribution of reaction time using Order-of-Processing (OP) diagrams. Journal of Mathematical Psychology, 27, 121-151.

Fisher, D. L. and Smith, J. M. (1987). Transient and equilibrium behavior of finite, tandem queues. Paper presented at the annual meetings of the Operations Research Society of America, St. Louis.

Goldstein, W. M. and Fisher, D. L. (1991). Stochastic networks as models of cognition: Derivation of response time distributions using Order-of-processing method. Journal of Mathematical Psychology, 35, 214-241.

Gross, D. and Harris, C. M. (1985). Fundamentals of queueing theory (2^{nd} edition). New York: John Wiley.

Rouse, W. B. (1980). Systems engineering models of human-machine interaction. New York: North Holland.

Sabol, M. A., Wisher, R. A. and Medici, C. A. (1991). Reaction time as a measure of skill development in copying Morse code. Proceedings of the Annual Meeting of the Military Testing Association. San Antonio, TX: U.S. Army.

Sternberg, S. (1969). Memory-scanning: Mental processes revealed by reaction time experiments. American Scientist, 57, 421-457.

Townsend, J. T. (in preparation). Initial mathematical models of early Morse code performance. U.S. Army Research Institute for the Behavioral and Social Sciences, Technical Report, Alexandria, VA.

Wisher, R. A., Kern, R. P., and Sabol, M. A. Cognitive process model for Morse intercept: Preliminary plan. Alexandria, VA: U.S. Army Research Institute. (Unpublished manuscript).

APPENDIX A

PARAMETERS OF GenCOPB, LimCOPB AND NoCOPB MODELS

GenCOPB Model. There are a total of 77 parameters in the GenCOPB model. To begin, consider the determination of the duration of the perception, recognition and execution processes. Thirty-six parameters are needed to describe the distributions of these three processes, only four of which are free. We assume that the time $T(r_i|t_i)$ that it takes to generate character r_i as a response, given tone sequence t_i was presented, is the sum of the time $T_p(e_i|t_i)$ that it takes to perceive a tone sequence t_i as element string e_i, given that tone sequence t_i was presented, plus the time T_r that it takes to recognize element string e_i plus the time T_e that it takes to organize and execute a response.

$$T(r_i|t_i) = T_p(e_i|t_i) + T_r + T_e.$$

Note that we assume that the recognition process is errorless and that the duration of this process does not vary from one element string to the next. Thus, we can drop both the indexed element string e_i which is output to the recognition process and the character code c_i which is generated from this process. Similarly, we assume that the execution process is errorless and that the duration of this process does not vary from one character code to the next. Thus, we can drop both the indexed character code c_i and the response r_i which is generated from the character code.

Since we assume that the duration T_r of the recognition process and duration T_e of the execution process do not depend on the identity of the stimulus which is being processed, we need only four parameters to describe the distribution of these durations: a shape parameter α_r and scale parameter β_r for the recognition stage and a shape parameter α_e and scale parameter β_e for the execution stage. These four parameters are free.

We assume that the distribution of $T_p(e_i|t_i)$ is a gamma with individual shape parameter shape parameters $\alpha_p(i)$ and common scale parameter β_p. Note that we could have individualized both the shape and scale parameters, but decided not to do so on this first attempt. The common scale parameter was selected so that the variance of the predicted observations was similar to that of the observed. The individual shape parameters were selected so that the equation, $\bar{T}(r_i|t_i) - E[T_r] - E[T_e] = \alpha_p(i)\beta_p$, was true, where $\bar{T}(r_i|t_i)$ is the average correct response latency (obtained from the confusion matrix), $E[T_r] = \alpha_r\beta_r$ and $E[T_e] = \alpha_e\beta_e$, i.e.,

$$\alpha_p(i) = \{\bar{T}(r_i|t_i) - E[T_r] - E[T_e]\}/\beta_p, \; i = 1,\ldots,31 \qquad \text{(A1.1)}$$

Thus, although there are 32 parameters [31 shape parameters, $\alpha_p(i)$, $i = 1,\ldots,31$, and one scale parameter, β_p) needed to describe the distributions of the durations of the 31 correct perception times, these parameters are all fixed by the confusion matrix data (for response times).

The next 10 parameters are free. To begin, consider the parameters needed to described the decay time in the perception, recognition, and execution buffers. We assume that the distribution of the decay time does not depend on the identity of the stimulus being processed or the fact that the stimulus is a letter as opposed to a period. Thus, we need a total of 6 parameters, 2 for the perception decay time (α_p^* and β_p^*), 2 for the recognition decay time (α_r^* and β_r^*) and two for the execution decay time (α_e^* and β_e^*). Next, we need three parameters for the capacity of the perception, recognition and execution buffers. We will label these γ_p, γ_r and γ_e. Finally, we will also need a parameter for the critical time which we will label δ. Again, these last ten parameters are free.

To complete the GenCOPB model, we need 31 parameters to determine when a character is perceived correctly and when it is perceived incorrectly as another character, given that the duration of the perception process is less

than the critical time δ. These parameters are fixed by the data in the confusion matrix (errors). Specifically, the probability $p(e_i|t_i)$ that element string e_i is generated at the perception stage, given that tone sequence t_i was presented and that the duration of the perception process was less than δ, is set equal to the observed probability $\hat{p}(r_i|t_i)$ that response r_i is made, given that tone sequence t_i was presented. The probability that element string e_j is generated at the perception stage, given that tone sequence t_i was presented ($i \neq j$) and that the duration of the perception process was less than the critical time δ, is simply the complement of the above, i.e., simply $1 - \hat{p}(r_i|t_i)$. In the simulation, the determination on each trial of whether a correct or incorrect element string was generated was made by sampling from a uniform [0,1] distribution. If the value sampled lay in the interval $[0,\hat{p}(r_i|t_i)]$, then the tone sequence t_i was transformed into the correct element string; otherwise, it was assumed that the element string was incorrect. No attempt was made to determine which of the incorrect element strings was produced.

Note that we do not require additional parameters to describe the output of the recognition and execution stages. This is because we assume that the probability $p(c_i|e_i)$ that character code c_i is generated by the recognition stage, given that element string e_i is input to the recognition stage, is equal to 1, as is the probability $p(r_i|c_i)$ that the execution stage generates response r_i, given that character code c_i was generated.

LimCOPB and NoCOPB Models. In order to reduce the number of free parameters, we made several simplifying assumptions. Specifically, we assumed that there was no room in the perception, recognition or execution buffers. And, therefore, the three free buffer size parameters and six free decay parameters were no longer necessary. Thus, the number of free parameters was reduced by nine, from 14 to 5. The number of fixed parameters remained at 63.

APPENDIX B:

ERROR AND LATENCY CONFUSION MATRICES

Below, we report the results from the 19 subjects we used to determine the relative fit of the LimCOPB and NoCOPB models. First, we list the data in the error confusion matrix. The first entry in row 1 (0.9095) is the probability that tone sequence t_1 was recalled as character r_1; the second entry (0) in row 1 is the probability that tone sequence t_1 was recalled as character r_2; and so on through the 41^{st} entry. The 42^{nd} entry in the i^{th} row is the probability that the i^{th} character is recalled correctly. For the first row, it is the probability (0.9095) that the first entry is recalled correctly. The 43^{rd} entry in the i^{th} row is the probability that a tone sequence is recalled as a character other than the one which is associated with it. In the first row, this probability, 0.0181, is equal to the product (7)(.0016). The 44^{th} entry in the i^{th} row is the probability of a period response to the i^{th}character. And the 45^{th} entry in the i^{th} row is the probability of that no response is made to the i^{th} character. Note that the first "row" includes the four lines indented underneath the first line; space did not permit the printing of all entries in a row of the confusion matrix as a single line:

.9095 0 0 0 0 .0016 0 0 .0016 0
.0016 0 0 .0016 .0016 0 0 .0016 0 .0016
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 .9095 .0181 .0280 .0444
0 .8109 .0016 .0329 .0016 .0016 0 .0033 0 0
0 0 0 0 .0016 0 0 .0016 0 0
0 .0049 0 .0066 0 .0033 0 .0016 0 0
0 0 0 0 0 0 0 0 0 0
0 .8109 .0608 .0987 .0296
0 0 .8793 0 0 0 0 0 0 0
.0046 .0015 0 .0031 0 0 0 0 0 0
.0015 0 0 .0077 .0046 0 .0046 0 0 0

0 0 0 0 0 0 0 0 0 0

0 .8793 .0294 .0387 .0526

0 .0280 0 .7747 0 .0016 0 0 0 0

.0066 .0016 .0033 .0066 0 0 0 .0082 .0049 0

.0033 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 .7747 .0657 .1349 .0247

.0016 0 0 0 .9079 0 0 0 .0115 0

0 0 0 0 0 0 0 0 0 .0016

0 0 0 0 0 .0016 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 .9079 .0263 .0247 .0411

0 .0017 .0017 0 .0017 .8591 0 0 0 0

0 .0085 0 .0017 0 0 0 .0034 .0034 0

0 .0017 0 0 0 0 0 .0068 0 0

.0068 0 0 0 0 0 0 0 0 0

0 .8591 .0441 .0730 .0238

0 .0015 0 .0046 0 .0046 .7895 0 0 0

.0015 0 .0015 .0015 .0093 .0046 0 0 0 0

.0077 .0015 .0015 0 0 .0015 0 0 0 .0108

0 0 0 0 0 0 0 0 0 0

0 .7895 .0603 .1161 .0341

0 .0032 0 0 0 0 0 .7703 0 .0016

0 .0016 0 0 .0016 0 0 0 .1069 0

.0016 .0048 .0016 0 0 .0016 0 0 .0016 0

0 0 0 0 0 0 0 0 0 0

0 .7703 .1276 .0606 .0415

.0064 0 0 0 .0128 0 0 0 .8692 0

0 0 0 .0128 .0016 0 0 0 .0032 0

0 .0016 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 .8692 .0446 .0415 .0447

.0016 .0016 0 .0016 0 0 0 0 .0016 .8273

.0016 0 0 0 .0016 .0132 0 0 .0016 0

0 0 .0115 .0016 0 0 0 0 .0049 .0033

.0033 0 0 0 0 0 0 0 0 0

0 .8273 .0526 .0954 .0247

.0016 0 .0016 .0016 0 0 0 .0016 .0016 0

.9298 .0064 0 .0016 0 0 0 .0032 0 0

0 0 0 .0016 .0016 0 .0032 0 0 0

0 0 0 0 0 0 0 0 0 0

0 .9298 .0256 .0207 .0239

0 0 0 0 0 .0016 .0016 0 0 0

.0016 .8581 0 0 .0016 .0080 0 .0048 0 0

0 0 0 0 0 0 0 .0064 0 0

.0016 0 0 0 0 0 0 0 0 0

0 .8581 .0319 .0558 .0542

0 0 0 0 0 0 0 .0033 .0016 0

.0016 0 .9030 .0033 .0099 0 0 0 0 0

0 0 0 0 .0016 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 .9030 .0279 .0461 .0230

0 0 0 0 0 0 0 0 0 0

0 0 .0016 .9260 0 0 0 .0033 .0016 0

0 .0033 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 .9260 .0148 .0444 .0148

0 .0016 0 0 0 0 .0033 .0016 0 .0016

0 .0016 .0033 0 .8766 .0016 .0016 0 .0033 .0016

0 0 0 0 0 0 0 0 .0197 .0033

0 0 0 0 0 0 0 0 0 0

0 .8766 .0461 .0395 .0378

0 0 0 0 0 .0066 .0016 0 0 .0082

0 0 0 0 0 .8059 0 0 0 0

0 0 .0016 .0033 .0016 0 0 .0033 .0016 .0066

.0016 0 0 0 0 0 0 0 0 0

0 .8059 .0461 .1250 .0230

.0016 0 0 0 0 0 .0016 0 0 0

0 0 0 0 0 0 .9059 .0016 0 0

0 0 .0016 .0032 .0080 .0064 0 0 .0016 0

0 0 0 0 0 0 0 0 0 0

0 .9059 .0287 .0447 .0207

.0033 0 0 .0016 0 .0099 0 0 0 0

.0049 .0099 0 0 0 0 0 .8586 0 0

0 0 .0082 0 0 0 0 .0016 0 0

0 0 0 0 0 0 0 0 0 0

0 .8586 .0476 .0609 .0329

.0016 0 0 .0016 0 0 0 .1003 .0214 0

0 0 .0033 0 .0016 0 0 0 .7763 0

.0082 0 0 0 0 0 .0016 0 0 0

0 0 0 0 0 0 0 0 0 0

0 .7763 .1415 .0526 .0296

.0016 0 0 0 .0099 0 0 0 0 0

0 0 .0033 0 0 0 0 .0016 .0016 .9309

0 0 0 0 .0016 0 0 .0016 0 0

0 0 0 0 0 0 0 0 0 0

0 .9309 .0231 .0296 .0164

.0033 0 0 .0066 0 .0016 .0016 0 0 .0033

0 0 0 0 .0016 0 0 0 .0033 0

.7549 .0526 .0099 0 0 0 0 0 0 0

.0049 0 0 0 0 0 0 0 0 0

0 .7549 .0921 .1283 .0247

0 .0033 0 0 0 .0016 0 0 0 0

0 0 0 0 .0016 0 .0033 0 .0016 0

.0263 .8372 0 .0016 0 0 0 0 0 .0016

.0016 0 0 0 0 0 0 0 0 0

0 .8372 .0444 .0839 .0345

.0033 0 0 0 .0016 0 .0016 0 0 0

0 0 .0016 0 .0016 0 0 0 0 0

.0049 .0016 .8651 0 0 0 0 0 0 0

.0016 0 0 0 0 0 0 0 0 0

0 .8651 .0214 .0872 .0263

0 .0033 .0016 .0016 .0016 0 0 .0016 0 0

.0033 .0016 0 0 0 .0016 0 0 0 0

0 .0016 0 .8355 .0033 .0033 .0016 .0148 0 0

.0016 0 0 0 0 0 0 0 0 0

0 .8355 .0461 .1020 .0164

0 .0016 .0049 0 0 0 0 0 0 .0016

.0066 .0016 0 0 .0016 .0016 .0049 0 0 .0016
.0066 .0016 0 .0049 .8684 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 .8684 .0411 .0625 .0280
.0049 .0016 0 .0016 0 0 .0099 0 0 0
0 0 0 0 .0016 .0033 .0066 0 0 0
.0016 0 0 .0033 0 .8306 0 .0016 0 .0049
.0049 0 0 0 0 0 0 0 0 0
0 .8306 .0493 .0806 .0395
.0033 0 .0016 0 0 0 0 0 0 0
.0033 .0082 0 0 .0016 0 0 0 0 0
0 0 0 0 0 .0016 .8717 0 0 .0016
0 0 0 0 0 0 0 0 0 0
0 .8717 .0279 .0609 .0395
0 0 0 .0016 .0099 .0378 .0016 0 0 0
0 .0230 0 0 0 .0033 0 0 .0016 0
0 .0016 .0016 .0016 0 0 .0016 .7615 0 .0016
.0049 0 0 0 0 0 0 0 0 0
0 .7615 .1036 .1053 .0296
0 .0017 0 0 0 0 0 .0085 0 .0102
0 0 0 0 .0204 0 0 0 .0017 0
0 0 .0017 0 0 0 0 0 .8846 .0187
0 0 0 0 0 0 0 0 0 0
0 .8846 .0627 .0289 .0238
.0016 .0016 0 .0016 0 0 .0016 .0016 0 .0064
0 0 0 0 .0064 .0064 0 .0016 0 0
0 0 0 0 0 0 0 0 .0064 .7911
.0016 0 0 0 0 0 0 0 0 0
0 .7911 .0430 .1260 .0399
0 0 0 0 0 .0115 0 0 0 .0016
0 .0016 0 0 0 0 0 0 0 .0016
.0099 0 .0115 .0016 .0049 .0016 0 0 0 0
.8651 0 0 0 0 0 0 0 0 0
0 .8651 .0477 .0658 .0214

Next, we present the results from the latency confusion matrix. Note that the only difference between these results and the results above is that there is not an entry for the latency of no response.

```
1048    0    0    0    0  369    0    0 2090    0
   779    0    0 1835  317    0    0  303    0 1926
     0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0
     0 1048 1088.4287 2330    0
0 1233 1001 1274 21193  224    0 1433    0    0
     0    0    0    0  311    0    0 2268    0    0
     0 1706    0 1248    0 2696    0 2501    0    0
     0    0    0    0    0    0    0    0    0    0
0 1233 1916.3907 1953    0
     0    0 1135    0    0    0    0    0    0    0
  1472 2384    0 1389    0    0    0    0    0    0
    92    0    0 1324 1418    0 2283    0    0    0
     0    0    0    0    0    0    0    0    0    0
     0 1135 1522.1196 2004    0
0 1261    0 1341    0 1101    0    0    0    0
  1347 1261 1047 1379    0    0    0 2621 3484    0
  1195    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0
     0 1341 1607.5070 2193    0
973    0    0    0 1111    0    0    0 1436    0
     0    0    0    0    0    0    0    0    0 1354
     0    0    0    0    0 1199    0    0    0    0
     0    0    0    0    0    0    0    0    0    0
     0 1111 1359.2394 1875    0
0 1916 1485    0 1007 1323    0    0    0    0
     0 1737    0 1243    0    0    0 1160  579    0
     0  747    0    0    0    0    0 1582    0    0
  1284    0    0    0    0    0    0    0    0    0
     0 1323 1364.7726 1874    0
0  887    0 1372    0 1241 1530    0    0    0
  1977    0 1737 1297 2008 1397    0    0    0    0
```

```
  1546 1045 1332    0    0 1482    0    0    0 1563
     0    0    0    0    0    0    0    0    0    0
     0 1530 1545.8831 1980    0
0  935    0    0    0    0    0 1147    0  349
     0 2404    0    0 1790    0    0    0 1213    0
   836 1011 4253    0    0 1426    0    0 1755    0
     0    0    0    0    0    0    0    0    0    0
     0 1147 1253.0952 2021    0
946    0    0    0 1213    0    0    0 1087    0
     0    0    0 1566 2082    0    0    0 1268    0
     0  827    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0
     0 1087 131.8751 2437    0
302 1951    0 2531    0    0    0    0 1023 1232
  2041    0    0    0 3988 1289    0    0 5981    0
     0    0  884 1524    0    0    0    0 1370 2249
  2014    0    0    0    0    0    0    0    0    0
     0 1232 161.3530 2169    0
1160    0 1481  941    0    0    0 1054 1535    0
  1077  961    0   81    0    0    0  877    0    0
     0    0    0 1258  820    0 2888    0    0    0
     0    0    0    0    0    0    0    0    0    0
     0 1077 1231.5001 1875    0
0    0    0    0    0  941  647    0    0    0
  1223 1084    0    0  888 1765    0 1253    0    0
     0    0    0    0    0    0    0 2225    0    0
  3048    0    0    0    0    0    0    0    0    0
     0 1084 166.6471 1818    0
0    0    0    0    0    0    0 2335 1287    0
   564    0 1201 1386 1207    0    0    0    0    0
     0    0    0    0 3047    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0
     0 1201 1505.4178 2559    0
0    0    0    0    0    0    0    0    0    0
     0    0  698 1167    0    0    0  880 1157    0
     0 2161    0    0    0    0    0    0    0    0
```

```
     0    0    0    0    0    0    0    0    0    0
     0 1167 1326.8674 1921    0
0  458    0    0    0    0 2049 1265    0 2195
     0 2919 1282    0 1083 1849 1830    0 2054 1050
     0    0    0    0    0    0    0    0 1855 1643
     0    0    0    0    0    0    0    0    0    0
     0 1083 1774.1836 1955    0
0    0    0    0    0 1597 5384    0    0 1960
     0    0    0    0    0 1308    0    0    0    0
     0    0 1841 1786 1532    0    0 1840 2885 1714
  2286    0    0    0    0    0    0    0    0    0
     0 1308 2004.8666 2006    0
1069    0    0    0    0    0 3025    0    0    0
     0    0    0    0    0    0 1169  999    0    0
     0    0  812 2869 2120 1234    0    0  695    0
     0    0    0    0    0    0    0    0    0    0
     0 1169 1742.1250 1716    0
738    0    0 1173    0 1744    0    0    0    0
  1211 1583    0    0    0    0    0 1220    0    0
     0    0  987    0    0    0    0 1123    0    0
     0    0    0    0    0    0    0    0    0    0
     0 1220 1347.0457 2344    0
1133    0    0  898    0    0    0 1514 1077    0
     0    0 1923    0 2122    0    0    0 1193    0
  1289    0    0    0    0    0  701    0    0    0
     0    0    0    0    0    0    0    0    0    0
     0 1193 1429.6855 1945    0
1018    0    0    0 2310    0    0    0    0    0
     0    0 1208    0    0    0    0 1035 1162 1195
     0    0    0    0  219    0    0  461    0    0
     0    0    0    0    0    0    0    0    0    0
     0 1195 156.7266 3177    0
1398    0    0 1808    0 1776 1848    0    0 1152
     0    0    0    0 2697    0    0    0 2705    0
  1605 1296 1700    0    0    0    0    0    0    0
  1828    0    0    0    0    0    0    0    0    0
```

```
     0 1605 1503.3225 2118    0
0 1162    0    0    0  511    0    0    0    0
     0    0    0    0 2019    0 1709    0  759    0
  1731 1177    0 1886    0    0    0    0    0 1576
  1966    0    0    0    0    0    0    0    0    0
     0 1177 1622.2775 1771    0
896    0    0    0 1546    0 3808    0    0    0
     0    0  983    0 2906    0    0    0    0    0
  1399 1203 1312    0    0    0    0    0    0    0
  1404    0    0    0    0    0    0    0    0    0
     0 1312 1616.3990 2166    0
0 1126 1447  867  870    0    0 2203    0    0
  1416 2125    0    0    0 1171    0    0    0    0
     0 2279    0 1358 3190 1606  943 1339    0    0
  3041    0    0    0    0    0    0    0    0    0
     0 1358 1602.5048 2055    0
0  957 1355    0    0    0    0    0    0 1706
   865 2372    0    0 2023 1146 1734    0    0 1522
  1730 3811    0 1992 1157    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0
     0 1157 1628.7236 2140    0
1558 1771    0  727    0    0 1776    0    0    0
     0    0    0    0 1901 1308 1699    0    0    0
  1986    0    0 1205    0 1338    0 2597    0 1892
  1975    0    0    0    0    0    0    0    0    0
     0 1338 1703.9824 2095    0
1274    0 1132    0    0    0    0    0    0    0
  1619 1516    0    0 2035    0    0    0    0    0
     0    0    0    0    0 1256 1303    0    0 1005
     0    0    0    0    0    0    0    0    0    0
     0 1303 1446.3634 1962    0
0    0    0  648 1293 1395  438    0    0    0
     0 1286    0    0    0 1901    0    0  765    0
     0  989 2412 2186    0    0 1948 1321    0  935
  1419    0    0    0    0    0    0    0    0    0
     0 1321 1361.5017 1740    0
```

```
0 1506    0    0    0    0    0 1131    0 1270
     0    0    0    0 1436    0    0    0 2570    0
     0    0 6043    0    0    0    0    0 1111 1514
     0    0    0    0    0    0    0    0    0    0
     0 1111 1548.1082 2201    0
1161 3424    0 1375    0    0 2036 1930    0 1591
     0    0    0    0 1283 1940    0  978    0    0
     0    0    0    0    0    0    0    0 1449 1373
  5232    0    0    0    0    0    0    0    0    0
     0 1373 179.7826 2146    0
0    0    0    0    0 1788    0    0    0 2000
     0 1278    0    0    0    0    0    0    0 1855
  1081    0 1384  902 1136 1043    0    0    0    0
  1216    0    0    0    0    0    0    0    0    0
     0 1216 1398.9321 1943    0
```

APPENDIX C:

THE FORTRAN SIMULATION CODE

```
C MOST RECENT UPDATE: 11/21/91 (APPROXIMATE)
C
C THIS PROGRAM, MORSE.FOR, WRITTEN 8/19/91 ON VAX AT UNIVERSITY OF
C    MASSACHUSETTS COLLEGE OF ENGINEERING FOR DR. ROBERT WISHER
C    BY DONALD L. FISHER.  THE PROGRAM SIMULATES THE BEHAVIOR OF
C    AN OPERATOR TRYING TO COPY MORSE CODE.
C
C THE UNDERLYING MODEL IS ONE IN WHICH THE FOLLOWING ASSUMPTIONS ARE MADE:
C
C    1) THERE ARE FOUR PROCESSES: ARRIVAL, AUDITORY PERCEPTION, CHARACTER
C    RECOGNITION AND MOTOR ORGANIZATION AND EXECUTION.  THE DURATION
C    OF THE ARRIVAL PROCESS IS A CONSTANT EQUAL TO THE DURATION
C    OF THE PARTICULAR MORSE CHARACTER THAT IS SENT.  THE DURATION OF THE
C    PERCEPTION, RECOGNITION AND EXECUTION PROCESSES ARE EACH GENERAL
C    GAMMA RANDOM VARIABLES.
C
C    2) THERE ARE FOUR BUFFERS: THE ARRIVAL BUFFER IS STIMBUF AND IS
C    UNLIMITED.  IT CONTAINS BOTH THE CHARACTER AND INTERCHARACTER
C    STIMULI.  THE PERCEPTION, RECOGNITION AND EXECUTION BUFFERS ARE
C    EACH SPECIFIED BY THE USER IN, RESPECTIVELY, INFO(2,2), INFO(2,3)
C    AND INFO(2,4).  NOTHING DECAYS OUT OF THE ARRIVAL BUFFER.  THE
C    DURATION OF THE DECAY PROCESSES IN THE THREE REMAINING BUFFERS
C    IS MODELED AS A GENERAL GAMMA RANDOM VARIABLE.
C
C    3) THE EXECUTION BUFFER HOLDS PERIODS, NOT CHARACTERS.  SPECIFICALLY,
C    IF A STIMULUS EXITS THE RECOGNITION STAGE AND THE EXECTUTION SERVER
C    IS FULL, THEN THE STIMULUS ENTERS THE EXECUTION BUFFER AS A PERIOD.
C    THE SIZE OF THE EXECTUTION BUFFER SHOULD BE KEPT RELATIVELY LARGE.
C    SIMILARLY, BECAUSE WE DO NOT WANT RAPID DECAY OF PERIODS, THE DURATION
C    OF A PERIOD IN THE EXECUTION BUFFER SHOULD BE SET RELATIVELY LARGE.
C
C
C
C
C DEFINE VARIABLES:
C
C    NCHR: NUMBER OF DIFFERENT CHARACTERS IN INPUT STREAM: 26 (JUST LETTERS),
C         31 (LETTERS + SPECIAL CHARACTERS), 41 (LETTERS, SPECIAL
C         CHARACTERS AND DIGITS); INITIALIZED IN SUBROUTINE SINV
C
C    NCHRBUF: DIMENSION OF CHRBUF; MUST BE GREATER THAN NGROUP*NREP
C
C    NCNT: SET EQUAL TO INFO(2,1) AT START; INFO(2,1) IS INITIALIZED
C         IN THE FILE MORSE.IN
C
C    NELEM: NEXT ELEMENT IN STIMULUS BUFFER (STIMBUF); READ IN FROM
C         SUBROUTINE SINV; ALWAYS START AT ELEMENT 0.
C
```

```
C     NGPM: GROUPS PER MINUTE; READ IN FROM SUBROUTINE SINV
C
C     NGROUP: NUMBER OF CHARACTERS IN A GROUP
C
C     NOUT: SET EQUAL TO 1 IF FULL OUTPUT (INFO,STATEC,STATED,STATEX,
C          STATEP,STATEY) DESIRED; OTHERWISE SET TO 0; READ IN FROM
C          SUBROUTINE SINV
C
C     NPER: THE NUMBER OF PERIOD RECODINGS PRODUCED WHEN A STIMULUS
C          LEAVES THE PERCEPTION SERVER.
C
C     NREP: NUMBER OF REPETITIONS OF A GROUP
C
C     NSTIM: NUMBER OF "STIMULI" = 2*NGROUP*NREP (NOTE THAT EVERY CHARACTER
C          IS FOLLOWED BY EITHER AN INTERCHARACTER BLANK OR AN INTER-
C          GROUP BLANK); NOTE THAT NELEM CAN REACH NSTIM AS AN UPPER
C          LIMIT
C
C     RCRIT: IF IT TAKES LONGER THAN RCRIT MILLISECONDS TO SERVICE A
C          STIMULUS IN THE PERCEPTION NODE, THEN THE STIMULUS IS
C          RECODED AS A PERIOD.
C
C
C
C
C DEFINE MATRICES:
C
C     ALPHA(2,4), BETA(2,4): IT IS ASSUMED THAT EACH SERVICE TIME AND EACH
C          DECAY TIME (EXCEPT ARRIVAL) IS A GAMMA RANDOM VARIABLE
C          WITH PARAMETERS ALPHA (a) AND BETA (b), I.E.,
C
C                              -a  a-1  -x/b
C          f(x) = [b   x    e     ]/gamma(a).
C
C          IN THIS CONTEXT, THE PARAMETERS FOR THE FOUR SERVICE AND
C          DECAY TIMES ARE:
C
C          ARRIVAL TIME :  ALPHA(1,1), BETA(1,1)  -- ASSUMED CONSTANT;
C               HOWEVER, COULD CHANGE TO GAMMA
C          ARRIVAL DECAY:  ALPHA(2,1), BETA(2,1)  -- ASSUMED UNLIMITED;
C               AGAIN, COULD CHANGE
C          PERCEP  TIME :  ALPHA(1,2), BETA(1,2)
C          PERCEP  DECAY:  ALPHA(2,2), BETA(2,2)
C          RECOG   TIME :  ALPHA(1,3), BETA(1,3)
C          RECOG   DECAY:  ALPHA(2,3), BETA(2,3)
C          EXEC    TIME :  ALPHA(1,4), BETA(1,4)
C          EXEC    DECAY:  ALPHA(2,4), BETA(2,4)
C
C     CHRBUF(410): PROGRAM WRITTEN TO HANDLE 50 GROUPS OF 5 CHARACTERS
C          EACH.  NOTE THAT STIMBUF HAS BOTH INTERACHARACTER AND
C          INTERGROUP BLANKS AND THUS SHOULD BE TWICE AS LARGE AS
C          CHARBUF.  ARRAY OF SIZE 410 BECAUSE EVERY 41 CHARACTERS
C          IS GUARANTEED TO CONTAIN DIGITS 1 - 41.  IF 31 CHARACTERS
C          IS USED, THEN CHRBUF IS FILLED TO POSITION 31*10 = 310.
```

```
C         IF 26 CHARACTERS ARE USED THEN ARRAY IS FILLED TO 26*10 = 260.
C         SINCE 260 IS GREATER THAN STANDARD 50*5 = 250 CHARACTERS,
C         I WENT WITH THE RATHER ODD 410.
C
C    CONMAT(41,45):
C
C         A) INITIALLY, IN ROW I, COLUMN J IS PROBABILITY THAT CHARACTER
C         I WAS SENT AND CHARACTER J WAS TYPED.  NOTE THAT PROBABILITIES
C         IN A ROW SHOULD ADD TO ONE.  HOWEVER, THIS DOES NOT MEAN THAT
C         AN ELEMENT IN ROW I IS NEVER MISSED OR TYPED AS A PERIOD.
C         THIS PROBLEM OCCURS LATER IN PROCESSES (CONMAT IS USED IN
C         THE PERCPETUAL BUFFER).  NOTE THAT IN ROW I AND COLUMN 44 I
C         CONDITIONAL PROBABILITY OF TYPING A PERIOD GIVEN CHARACTER I
C         AND IN ROW I AND COLUMN 45 IS CONDITIONAL PROBABILITY OF
C         TYPING NOTHING GIVEN CHARACTER I.  THE OUTPUT INFORMATION
C         ON PERIODS AND NO RESPONSES IS CONTAINED IN KILL: THE NUMBER
C         OF ENTRIES IN THE PERCEPTION AND RECOGNITION NODES IS THE
C         NUMBER OF NO RESPONSES; THE NUMBER OF ENTRIES IN THE
C         EXECUTION NODE IS THE NUMBER OF PERIODS.
C
C         B) AFTER READING IN CONMAT, IF P(J|I) IS THE PROBABILITY THAT
C         J IS TYPED WHEN I IS PRESENTED, THEN IN ROW I, COLUMN 1 WE
C         PLACE P(1|I), IN ROW I, COLUMN 2 WE PLACE P(1|I) + P(2|I),
C         AND SO ON.
C
C    CONFUS(41,41): IN ROW I AND COLUMN J IS THE NUMBER OF TIMES IN THE
C         SIMULATION THAT                          CHARACTER J REPLACES
CHARACTER I AT THE
C         END OF SERVICE ON THE PERCEPTION NODE.  THIS INFORMATION IS
C         OBTAINED FROM CONMAT.
C
C    DATERR(41,45): ERROR DATA COMES FROM SUBJECTS.  IN ROW I AND COLUMN J
C         IS PROBABILITY THAT A SUBJECT RESPONDS WITH STIMULUS J
C         GIVEN SUBJECT HAS BEEN PRESENTED STIMULUS I.  I = 1,...,41
C         SINCE NUMBER OF STIMULI CAN GO UP TO 41.  J = 1,...,41,
C         44,45.  J = 44 IS PERIOD RESPONSE.  J = 45 IS NO RESPONSE.
C         J = 42,43 ARE USED AS STIMULUS VALUES (INTERCHARACTER AND
C         INTERGROUP BLANKS), NOT RESPONSES.  HOWEVER, IN COLUMN 42
C         I'VE PLACED THE PERCENTAGE CORRECT RESPONSES, I.E., IN
C         DATERR(IA,42) I'VE PLACED DATERR(IA,IA).  AND IN COLUMN
C         43 I'VE PLACED THE INCORRECT CHARACTER ONLY ERRORS.
C         THUS, THE PROBABILITIES SUMMED WITHIN A ROW ACROSS
C         COLUMNS 42 - 45 OUT TO EQUAL 1.
C
C    INFO(2,4): CONTAINS INFORMATION ON PARAMETERS OF ARRIVAL, PERCEPTION,
C         RECOGNITION AND EXECUTION PROCESSES IN, RESPECTIVELY,
C         COLUMNS 1, 2, 3 AND 4:
C
C         INFO(1,1) = NUMBER OF CHARACTERS ON ARRIVAL SERVER (1 OR 0)
C         INFO(2,1) = TOTAL NUMBER OF ARRIVALS IN SESSION LEFT; NOTE
C                     THAT ARRIVALS INCLUDE INTERCHARACTER AND INTER-
C                GROUP INTERVALS
C
C         INFO(1,2) = NUMBER OF CHARACTERS ON PERCEPTION SERVER AND
```

```
C                AND IN PERCEPTION BUFFER (E.G., IF THERE IS ONE
C                CHARACTER CURRENTLY BEING PERCEIVED AND TWO
C                CHARACTERS IN THE BUFFER, THEN INFO(1,2) = 3)
C          INFO(2,2) = MAXIMUM NUMBER OF CHARACTERS WHICH CAN BE
C                SERVICED AND BUFFERED FOR PERCEPTION
C
C          INFO(1,3) = NUMBER OF CHARACTERS ON EXECUTION SERVER AND IN
C                RECOGNITION BUFFER.
C          INFO(2,3) = MAXIMUM NUMBER OF CHARACTERS WHICH CAN BE
C                SERVICED AND BUFFERED FOR RECOGNITION.
C
C          INFO(1,4) = NUMBER OF CHARACTERS ON EXECUTION SERVER AND IN
C                EXECUTION BUFFER.
C          INFO(2,4) = MAXIMUM NUMBER OF CHARACTERS WHICH CAN BE
C                SERVICED AND BUFFERED FOR EXECUTION.
C
C     INTERC(15),INTERG(15): CONTAINS IN COLUMN I THE INTERCHARACTER
C          (INTERGROUP) INTERVAL FOR RATES OF, RESPECTIVELY, 6,8,10,12
C          14,16,18,20 GROUPS PER MINUTE.
C
C     KEEP(41,46): CONTAINS NUMBER OF TIMES IN KEEP(I,J) CHARACTER J IS
C          TYPED WHEN CHARACTER I WAS PRESENTED.  NOTE THAT ENTRIES
C          IN KEEP(I,J) CAN BE NO GREATER THAN ENTRIES IN CONFUS(I,J)
C          BECAUSE CHARACTERS CAN DECAY AT A NODE OR NOT GAIN ENTRY
C          TO A NODE AFTER THEY ARE CONFUSED IN THE PERCPETION SERVER.
C          NOTE THAT BEFORE SUBROUTINE SOUT, KEEP(IA,42),KEEP(IA,43)
C          AND KEEP(IA,45) ARE BLANK; KEEP(IA,44) CONTAINS THE PERIOD
C          COUNT.  SUBROUTINE SOUT PUTS INTO KEEP(IA,42) THE ENTRY
C          IN KEEP(IA,IA), I.E., THE NUMBER OF CORRECT RESPONSE. IT
C          PUTS INTO KEEP(IA,43) THE NUMBER OF INCORRECT CHARACTER
C          CHARACTER RESPONSES.  AND IT PUTS INTO KEEP(IA,45) THE NUMBER
C          OF NO RESPONSES (BY SUMMING OVER KILL(IA,J),J=1,8).
C
C     KEEPSP(5): KEEPSP(I) CONTAINS NUMBER OF CHARACTERS IN ITH POSITION
C          IN A GROUP WHICH MADE IT THROUGH EXECUTION STATE.  YIELDS
C          STANDARD SERIAL POSITION CURVES.  ONLY CORRECT CHARACTERS
C          ARE COUNTED.
C
C     KILL(41,8): CONTAINS NUMBER OF TIMES CHARACTER IN ROW IS LOST
C          BECAUSE OF BUMPOUT, DECAY OR PERIOD.  COLUMNS 1 - 4 CONTAIN
C          ARRIVAL, PERCEPTION, RECOGNITION AND EXECUTION BUMPOUT;
C          NEXT 4 COLUMNS CONTAIN DECAY.  FOR EXAMPLE, IF KILL(11,3) = 4,
C          THIS MEANS THAT CHARACTER 11 ON LEAVING THE PERCEPTION SERVER
C          HAS BEEN BUMPED 4 TIMES BECAUSE THE RECOGNITION BUFFER IS
C          FULL.  IF KILL(21,6)=5, THIS MEANS THAT CHARACTER 21 ON THE
C          PERCEPTION BUFFER HAS DECAYED 5 TIMES BECAUSE THE PERCEPTION
C          SERVER IS FULL.  NOTHING IS BUMPED OUT FROM THE ARRIVAL
C          QUEUE NOR DOES ANY DECAY OCCUR.  FURTHERMORE, NOTHING IS
C          BUMPED OUT OF THE RECOGNITION BUFFER NOR DOES DECAY WITH
C          COMPLETE LOSS OCCUR.  RATHER, PERIOD IS TYPED.  THUS,
C          IF KILL(10,4) = 6, THIS MEANS THAT SIX PERIODS WERE
C          TYPED BECAUSE THE CHARACTER BUFFER SPACE IN THE EXECUTION
C          BUFFER WAS EXCEEDED.  IF KILL(10,8) = 2, THIS MEANS THAT
C          TWO PERIODS WERE TYPED BECAUSE THE DECAY TIME IN THE
```

```
C          EXECUTION BUFFER WAS EXCEEDED.
C
C     DATLAT(41,45): CONTAINS RESPONSE TIMES FROM SUBJECTS.  LIKE MATRIX
C          ERROR OTHERWISE EXCEPT FOR NO ENTRY IN COLUMN 45 (SINCE
C          NO TIMES FOR NO RESPONSES).
C
C     STATEC(30,4),STATED(30,4): COLUMN 1 CONTAINS INFORMATION ON THE
C          ARRIVAL PROCESS; COLUMN 2 CONTAINS INFORMATION ON THE
C          PERCEPTION PROCESS; COLUMN 3 CONTAINS INFORMATION ON
C          THE RECOGNITION PROCESS; AND COLUMN 4 CONTAINS INFORMATION
C          ON EXECUTION PROCESS IN BOTH STATEC AND STATED.
C
C          STATEC(1,1) = MORSE CHARACTER CURRENTLY ARRIVING
C          STATED(1,1) = REMAINING DURATION OF MORSE CHARACTER
C
C          STATEC(1,2) = MORSE CHARACTER CURRENTLY BEING PERCEIVED; NOTE
C                        THAT THIS IS NOT NECESSARILY THE CHARACTER WHICH
C                        IS SENT IF THERE IS A CONFUSION
C          STATED(1,2) = REMAINING DURATION OF PERCEPTION PROCESS
C          STATEC(2,2) = MORSE CHARACTER IN PERCEPTION BUFFER (IF ANY)
C          STATED(2,2) = REMAINING DECAY TIME OF ABOVE CHARACTER
C          STATEC(3,2) = ADDITIONAL MORSE CHARACTER IN PERCEPTION
C                        BUFFER (IF ANY)
C          STATED(3,2) = REMAINING DECAY TIME OF THIS MORSE CHARACTER
C          (AND SO ON FOR EACH ADDITIONAL MORSE CHARACTER IN PERCEPTION
C          BUFFER)
C
C          STATEC(1,3) = MORSE CHARACTER CURRENTLY BEING RECOGNIZED
C          STATED(1,3) = REMAINING DURATION OF RECOGNITION PROCESS
C          STATEC(2,3) = MORSE CHARACTER IN RECOGNITION BUFFER (IF ANY)
C          STATED(2,3) = REMAINING DECAY TIME OF ABOVE CHARACTER
C          STATEC(3,3) = ADDITIONAL MORSE CHARACTER IN RECOGNITION
C                      BUFFER (IF ANY)
C          STATED(3,3) = REMAINING DECAY TIME OF THIS MORSE CHARACTER
C          (AND SO ON FOR EACH ADDITIONAL MORSE CHARACTER IN RECOGNITION
C          BUFFER)
C
C          STATEC(1,4) = MORSE CHARACTER CURRENTLY BEING RESPONDED TO
C          STATED(1,4) = REMAINING DURATION OF ABOVE CHARACTER
C          STATEC(2,4) = MORSE CHARACTER IN RESPONSE BUFFER (IF ANY)
C          STATED(2,4) = REMAINING DECAY TIME OF ABOVE CHARACTER
C          (AND SO ON)
C
C     STATEP(30,4): CONTAINS IN ROW I AND COLUMN J POSITION OF CHARACTER
C          IN STREAM.  FOR EXAMPLE, IF THE GROUPS ARE OF SIZE 5
C          AND THE CHARACTER CURRENTLY ON THE RECOGNITION SERVER IS
C          THE SECOND CHARACTER IN A STREAM, THEN STATEP(1,2) = 2.
C
C     STATEX(30,4),STATEY(30,4): STATEX IS INDENTICAL TO STATEC EXCEPT
C          FOR THE FACT THAT THERE IS NO CHANGE TO THE IDENTITY OF
C          A CHARACTER MISPERCIEVED.  THIS MAKES IT POSSIBLE TO TRACK
C          THE RESPONSE TIME TO THE CHARACTER WHICH WAS SENT.  STATEY
C          KEEPS TRACK OF THE TOTAL TIME THAT A CHARACTER SPENDS IN
C          THE SYSTEM.  MEASUREMENT STARTS FROM THE POINT WHERE THE
```

```
C          CHARACTER HAS ARRIVED (I.E., MEASUREMENT STARTS WHEN THE
C          CHARACTER ENTERS THE PERCEPTION QUEUE OR SERVER) AND
C          CONTINUES UNTIL THE RESPONSE HAS BEEN EXECUTED.
C
C     STIMBUF(820):  STIMBUF CONTAINS THE CHARACTER, INTERCHARACTER CODE OR
C          INTERGROUP CODE.  THE SPACE BETWEEN CHARACTERS IS IDENTIFIED
C          BY THE                                  INTEGER 42; THE SPACE BETWEEN
GROUPS
C          IS IDENTIFIED BY THE INTEGER 43.  THE FIRST CHARACTER APPEARS
C          IN STIMBUF(1), THE SECOND CHARACTER IN STIMBUF(2), AND SO ON.
C          NOTE THAT STIMBUF MUST BE TWICE THE SIZE OF CHRBUF.
C
C     STIMDUR(43): CONTAINS DURATIONS OF 1 - 26 LETTERS, 5 SPECIAL
C          CHARACTERS, 10 DIGITS AND BOTH INTERCHARACTER (42) AND
C          INTERGROUP (33) INTERVALS.  NOTE THAT DURATION OF
C          INTERCHARACTER AND ITNERGROUP INTERVALS WILL VARY WITH GPM.
C          THIS WILL SUPERCEDE WHATEVER IS READ INTIALLY INTO
C          STIMDUR.
C
C     TIME(41,45): CONTAINS IN TIME(I,J) FOR EVERY TIME CHARACTER I
C          IS SENT AND CHARACTER J IS TYPED A RUNNING SUM OF THE
C          RESPONSE TIME.  AT THE END, IN SUBROUTINE SOUT,
C          THE AVERAGE IS COMPUTED IN TIME BY DIVIDING TIME(I,J)
C          BY KEEP(I,J).
C
C     TIMESP(5): TIMESP(I) CONTAINS RESPONSE TIMES OF ALL CORRECTLY
C          RECOGNIZED CHARACTERS FROM POSITION I IN A GROUP.  AT END
C          TIMESP YIELDS AVERAGE.  TOTAL OBSERVATIONS FOR ITH CHARACTER
C          IN STREAM IS KEPT IN KEEPSP(I).
C
C
C
C
C DEFINE FUNCTIONS
C
C     SGAMMA(...): GENERATES PSEUDO RANDOM
C          GAMMA RANDOM VARIABLE USING ROUTINE IN AVERILL M. LAW AND
C          W. DAVID KELTON, SIMULATION MODELING AND ANALYSIS,
C          NEW YORK, MCGRAW HILL, 1982, PAGE 257.
C
C
C
C
C DEFINE SUBROUTINES:
C
C     SARR(...): THIS CREATES NEXT STATE GIVEN MINIMUM HAS OCCURRED ON
C          ARRIVAL SERVER. ACTIONS MUST BE TAKEN TO THE INFO AND
C          STATE MATRICES FOR BOTH THE ARRIVAL AND PERCEPTION PROCESSES:
C          A-S)  IF THERE ARE MORE ARRIVALS CHANGE STATEC(1,1) TO
C                STIMBUF(NELEM); CHANGE STATED(1,1) TO RANDOM DURATION.
C                OTHERWISE STATEC(1,1) = STATED(1,1) = 0.
C          A-I)  IF THERE ARE MORE ARRIVALS DECREMENT INFO(2,1);
C                OTHERWISE SET INFO(1,1) TO ZERO.
C          P-S)  IF ARRIVAL IS A CHARACTER (NOT A BLANK TIME) AND
```

```
C              IF THERE IS ROOM IN THE PERCEPTION BUFFER THEN
C              SET STATEC(INFO(1,2)+1,2) = OLD STATEC(1,1); SET
C              STATED(INFO(1,2)+1,2) = RANDOM DURATION.  IF
C              ARRIVAL IS A CHARACTER AND NO ROOM IN RECOGNITION
C              BUFFER, PUT DISPLACED CHARACTER IN KILL DEPOSITORY
C         P-I) IF ARRIVAL IS A CHARACTER AND IF THERE IS ROOM IN THE
C              PERCEPTION BUFFER, THEN INFO(1,2) = INFO(1,2) + 1
C
C
C     SCHRBUF(...): PUTS INTO CHRBUF(410) RANDOM PERMUTATIONS
C          OF DIGITS 1 - 41 (CORRESPONDING TO IDENTIFIERS FOR MORSE
C          CHARACTERS, INCLUDING SPECIAL CHARACTERS)
C
C     SEXEBUF(...): PERFORMS SAME OPERATIONS AS
C          SPERBUF IN EXECUTION BUFFER
C
C     SGPM(...): PUTS INTO STIMDUR(42) THE INTERCHARACTER
C          DURATION FROM INTERC(I) WHERE INTERC(1) IS 6 GPM AND
C          INTERC(15) IS 20 GPM.  SIMILARLY FOR STIMDUR(43) AND
C          INTERG(J).
C
C     SMINI(...):  IDENTIFIES CHARACTER WITH MINIMUM
C          REMAINING PROCESSING TIME AND SETS KROW AND KCOL TO THE
C          ROW AND COLUMN ENTRY IN STATE OF THE DURATION OF THIS
C          CHARACTER.  FOR EXAMPLE, SUPPOSE:
C
C          STATED(1,1) = 50, STATED(1,2) = 100, STATED(2,2) = 110,
C          STATED(1,3) = 200, STATE(1,3) = 120 AND STATE(1,4) = 80
C
C          THEN KROW = 1 AND KCOL = 1 SINCE 50 IS THE MINIMUM DURATION.
C
C     SPER(...):  IN STATES WHERE THE MINIMUM DURATION IS IN THE
C          PERCEPTUAL SERVER, SPER MAKES THE NECESSARY CHANGES.  NOTE
C          THIS IS WHERE A PERIOD IS INTRODUCED.  SPECIFICALLY,
C          IF THE TIME IT TAKES TO PERCEIVE A STIMULUS IS LONGER
C          THAN SOME TIME RCRIT, THEN THE STIMULUS IS CODED AS
C          A PERIOD.  OTHERWISE (I.E., IF THE SERVICE TIME IS
C          LESS THAN THE CRITICAL TIME) THEN WE USE THE CONFUSION
C          MATRIX TO DETERMINE WHETHER THE CHARACTER IS CODED
C          CORRECTLY OR INCORRECTLY.
C
C     SPERBUF(...): IN STATES WHERE THE MINIMUM DURATION
C          IS IN THE PERCEPTUAL BUFFER, SPERBUF REMOVES THE
C          MINIMUM DURATION PROCESS FROM STATEC AND STATED AND MOVES
C          EVERYTHING ABOVE IT UP ONE IN THE QUEUE.  FOR EXAMPLE, SUPPOSE
C          WE HAVE FIVE ITEMS IN THE PERCEPTUAL BUFFER AND ONE ON THE
C          SERVER.  WE APPLY SMINI AND OBTAIN:
C
C          STATED(1,2) = 50, STATED(2,2) = 150, STATED(3,2) = 0,
C          STATED(4,2) = 170, STATE(5,2) = 120 AND STATE(6,2) = 130.
C
C          SPERBUF MOVES THE THIRD (STATED(4,2)), FOURTH (STATED(5,2))
C          AND FIFTH (STATED(6,2)) UP ONE AND REPLACES STATE(6,2) WITH 0,
C          SO THAT WE HAVE:
```

```
C
C          STATED(1,2) = 50, STATED(2,2) = 150, STATED(3,2) = 170,
C          STATED(4,2) = 120, STATE(5,2) = 130 AND STATE(6,2) = 0.
C
C          NOTE THAT THE SAME CHANGES ARE MADE TO STATEC
C
C     SPERERR(JTEMPC,CONMAT,CONFUS,SEED): LOOKS AT CONFUSION MATRIX AND DECIDES
C          WHETHER TO REPLACE CURRENT CHARACTER WHICH HAS JUST BEEN
C          PERCEIVED WITH ANOTHER CHARACTER.  FOR EXAMPLE, SUPPOSE THAT
C          THERE WERE FOUR CHARACTERS.  SET P(J|I) EQUAL TO THE
C          PROBABILITY OF PERCEIVING J GIVEN I.  SUPPOSE P(1|1) = .4,
C          P(2|1) = .3, P(3|1) = .2 AND P(4|1) = .1.  THEN, IN CONMAT
C          IN ROW 1 WE CODE .4, .7, .9 AND 1.0.  WE THEN GENERATE
C          A VALUE FOR A UNIFORM[0,1] RANDOM VARIABLE AND DETERMINE
C          WHETHER THIS VALUE IS IN THE INTERVAL [0,.4).  IF SO,
C          THEN THE CHARACTER 1 IS CORRECTLY CODED AS A 1.  IF NOT,
C          THEN WE DETERMINE WHETHER THE VALUE IS IN THE INTERVAL
C          [.4,.7).  IF SO, THEN A 2 IS INCORRECTLY CODED AS A 1.
C          AND SO ON.
C
C     SRECBUF(STATEC,STATED,INFO,KROW,KCOL,KILL): PERFORMS SAME OPERATIONS AS
C          SPERBUF IN RECOGNITION BUFFER
C
C     SSTIM(CHRBUF,STIMBUF,NGROUP,NREP,NSTIM): PUTS FIRST CHARACTER IN CHRBUF
C          INTO STIMBUF, THEN INTERCHARACTER IDENTIFIER, THEN NEXT
C          CHARACTER IN CHRBUF, AND SO ON UNTIL NGROUP CHARACTERS HAVE
C          BEEN PLACED IN STIMBUF, AT WHICH POINT AN INTERGROUP
C          IDENTIFIER IS PLACED IN STIMBUF.  THIS IS REPEATED FOR
C          EACH SUCCEEDING GROUP.
C
C     SSUB(STATEC,STATED,INFO,KROW,KCOL): SUBTRACTS THE MINIMUM DURATION OF THE
C          PROCESS FOUND IN SMIN FROM THE DURATION OF EACH OF THE
C          PROCESSES IN STATED, LEAVING THE REMAINING DURATION FOR EACH
C          PROCESS.  FOR EXAMPLE, AFTER SSUB OPERATION ON THE OUTPUT
C          OF SMIN ABOVE, WE WOULD HAVE:
C
C          STATED(1,1) = 0, STATED(1,2) = 50, STATED(2,2) = 60,
C          STATED(1,3) = 150, STATE(1,3) = 120 AND STATE(1,4) = 30.
C
C
C
C
C TYPE VARIABLES
      INTEGER NCHR
      INTEGER NCHRBUF
      INTEGER NGPM
      INTEGER NCNT
      INTEGER KROW,KCOL          !ROW AND COLUMN MARKERS OF MIN DUR IN STATE
      INTEGER NELEM
      INTEGER NGROUP,NREP,NSTIM
      INTEGER NOUT
      INTEGER NPER
      INTEGER*4 SEED
      REAL RCRIT
```

```
C
C
C
C
C TYPE MATRICES
      INTEGER CHRBUF(410)
      INTEGER CONFUS(41,41)
      INTEGER INFO(2,4)
      INTEGER KEEP(41,46)
      INTEGER KEEPSP(5)
      INTEGER KILL(41,8)
      INTEGER STATEC(30,4)
      INTEGER STATEP(30,4)
      INTEGER STATEX(30,4)
      INTEGER STIMBUF(820)
      REAL ALPHA(2,4),BETA(2,4)
      REAL CONMAT(41,45)
      REAL DATERR(41,45),DATLAT(41,45)
      REAL INTERC(15),INTERG(15)
      REAL STATED(30,4)
      REAL STATEY(30,4)
      REAL STIMDUR(43)
      REAL TIME(41,45)
      REAL TIMESP(5)
C
C
C
C
C DATA INITIALIZATION STATEMENTS
      DATA SEED/98753/
      DATA CONFUS/1681*0/
      DATA INFO/8*0/
      DATA INTERC/1260.,1030.,860.,729.,624.,537.,466.,405.,353.,
      C              308.,268.,233.,203.,175.,150./
      DATA INTERG/2940.,2402.,2008.,1701.,1455.,1254.,1087.,
      C              945.,824.,718.,626.,545.,473.,408.,350./
      DATA KEEP/1886*0/
      DATA KEEPSP/5*0/
      DATA KILL/328*0/
      DATA STATEC/120*0/
      DATA STATED/120*0./
      DATA STATEP/120*0/
      DATA TIME/1845*0./
      DATA TIMESP/5*0./
C
C OPEN FILES
      OPEN(UNIT=1,FILE='MORSE.IN',STATUS='OLD',READONLY)
      OPEN(UNIT=2,FILE='MORSE.OUT',STATUS='NEW')
      OPEN(UNIT=3,FILE='MORSE.DAT',STATUS='OLD',READONLY)
C
C READ IN MATRIX INFORMATION: SIMULATION AND DATA
      CALL SINP(STATEC,STATED,INFO,STIMDUR,ALPHA,BETA,
      C            STATEX,STATEY)
      CALL SDAT(DATERR,DATLAT,CONMAT)
```

```
C
C BOUNCE BACK HERE FOR SIMULATION USING NEW GPM
99    CONTINUE
C
C INITIALIZE VARIABLES
      CALL SINV(INFO,NCHR,NCHRBUF,NGPM,NGROUP,NREP,NSTIM,NELEM,NCNT,
      C       NOUT,NPER,RCRIT,KCOPB,SEED)
C
C RANDOMIZE INPUT STREAM OF CHARACTERS
      CALL SCHRBUF(SEED,CHRBUF,NCHR,NOUT)
C
C INSERT INTERCHARACTER AND INTERGROUP IDENTIFIERS INTO STREAM
      CALL SSTIM(CHRBUF,STIMBUF,NGROUP,NREP,NSTIM,NOUT)
C
C SET INTERCHARACTER AND INTERGROUP DURATION FOR A PARTICULAR RATE (GROUPS
C     PER MINUTE)
      CALL SGPM(STIMDUR,INTERC,INTERG,NGPM)
C
C BEGIN CREATING STATES
100   CONTINUE                              !BOUNCE BACK HERE IF MORE STATES
C     NO MORE STATES IF NOTHING ACTIVE ON SERVERS
      IF((INFO(1,1)+INFO(1,2)+INFO(1,3)+INFO(1,4)).EQ.0) GOTO 999
C
C OBTAIN MINIMUM PROCESSING DURATION
      CALL SMINI(STATEC,STATED,INFO,KROW,KCOL,NOUT)
C
C SUBTRACT MINIMUM PROCESSING DURATION
      CALL SSUB(STATEC,STATED,INFO,KROW,KCOL)
C
C IF ARRIVAL COMPLETES FIRST THEN:
      IF(KCOL.EQ.1) THEN
           NELEM = NELEM + 1
           CALL SARR(STATEC,STATED,INFO,KROW,KCOL,STIMBUF,NELEM,
      C          KILL,SEED,STIMDUR,ALPHA,BETA,STATEX,STATEY,STATEP,
      C          NOUT,DATLAT,KCOPB)
           GO TO 100
      END IF
C
C IF CHARACTER ON PERCEPTION SERVER COMPLETES FIRST THEN:
      IF((KCOL.EQ.2).AND.(KROW.EQ.1)) THEN
           CALL SPER(STATEC,STATED,INFO,KROW,KCOL,STIMBUF,NELEM,
      C               KILL,CONMAT,CONFUS,SEED,ALPHA,BETA,STATEX,STATEY,
      C               STATEP,NOUT,NPER,RCRIT,DATLAT)
           GO TO 100
      END IF
C
C IF CHARACTER IN PERCEPTION BUFFER COMPLETES FIRST, THEN
C     SQUASH PERCEPTION BUFFER
      IF((KCOL.EQ.2).AND.(KROW.GT.1)) THEN
           CALL SPERBUF(STATEC,STATED,INFO,KROW,KCOL,KILL,STATEX,
      C                         STATEY,STATEP,NOUT)
           GO TO 100
      END IF
C
```

```
C IF CHARACTER ON RECOGNITION SERVER COMPLETES FIRST THEN:
      IF((KCOL.EQ.3).AND.(KROW.EQ.1)) THEN
           CALL SREC(STATEC,STATED,INFO,KROW,KCOL,STIMBUF,NELEM,
     C               KILL,SEED,ALPHA,BETA,STATEX,STATEY,STATEP,NOUT,
     C               DATLAT)
           GO TO 100
      END IF
C
C IF CHARACTER IN RECOGNITION BUFFER COMPLETES FIRST, THEN
C     SQUASH RECOGNITION BUFFER
      IF((KCOL.EQ.3).AND.(KROW.GT.1)) THEN
           CALL SRECBUF(STATEC,STATED,INFO,KROW,KCOL,KILL,STATEX,
     C               STATEY,STATEP,NOUT)
           GO TO 100
      END IF
C
C IF CHARACTER ON EXECUTION SERVER COMPLETES FIRST THEN:
      IF((KCOL.EQ.4).AND.(KROW.EQ.1)) THEN
           CALL SEXE(STATEC,STATED,INFO,KROW,KCOL,STIMBUF,NELEM,
     C          KEEP,KILL,ALPHA,BETA,STATEX,STATEY,TIME,STATEP,KEEPSP,
     C          TIMESP,NOUT,DATLAT)
           GO TO 100
      END IF
C
C IF CHARACTER IN EXECUTION BUFFER COMPLETES FIRST, THEN
C     SQUASH EXECUTION BUFFER
      IF((KCOL.EQ.4).AND.(KROW.GT.1)) THEN
           CALL SEXEBUF(STATEC,STATED,INFO,KROW,KCOL,KILL,
     C          STATEX,STATEY,STATEP,NOUT)
           GO TO 100
      END IF
C
C GET HERE WHEN NO MORE STATES
999   CONTINUE
C
C
C WRITE SELECTED OUTPUT
      CALL SOUT(INFO,KEEP,KEEPSP,KILL,CONFUS,TIME,TIMESP,NCHR,NCNT,
     C            NOUT,NGPM,NPER,DATERR,DATLAT,RCRIT,KCOPB)
C
C ZERO INPUT TO START SIMULATION AT NEW RATE
      CALL SZERO(CONFUS,NCHR,INFO,NCNT,KEEP,KEEPSP,KILL,STATEC,
     C       STATEP,STATEX,STATED,STATEY,TIME,TIMESP,NPER)
      GO TO 99

C
C END MAIN
      END
```

```
                                SUBROUTINE SINV
C
C
C
C
C
      SUBROUTINE SINV(INFO,NCHR,NCHRBUF,NGPM,NGROUP,NREP,NSTIM,NELEM,
      C                    NCNT,NOUT,NPER,RCRIT,KCOPB,SEED)
      INTEGER KCOPB
      INTEGER NCHR
      INTEGER NCHRBUF
      INTEGER NCNT
      INTEGER NGPM
      INTEGER NELEM
      INTEGER NGROUP,NREP,NSTIM
      INTEGER NOUT
      INTEGER NPER
      INTEGER*4 SEED
      INTEGER INFO(2,4)
      REAL RCRIT
8     FORMAT(' ENTER 1 TO CONTINUE, 0 TO STOP    = ',$)
10    FORMAT(' ENTER ELEMENT IN STREAM: 0        = ',$)
12    FORMAT(' ENTER GROUPS PER MINUTE: 6 - 20   = ',$)
14    FORMAT(' DO YOU WANT FULL OUTPUT: 1/0      = ',$)
16    FORMAT(' ENTER CRITICAL VALUE IN MS:       = ',$)
18    FORMAT(' ENTER 1 FOR COPY BEH; 0 NO COPB   = ',$)
C     BEGIN MAIN BODY OF PROGRAM
      WRITE(5,*)
      WRITE(5,8)
      READ(5,*) NSTOP
      IF(NSTOP.EQ.0) THEN
           WRITE(5,*)
           WRITE(5,*) ' OUTPUT IN FILE MORSE.OUT '
           WRITE(5,*)
           STOP
      END IF
      JNCNT = (INFO(2,1) -1)/2 + 1
      WRITE(5,*)
      WRITE(5,*) '    NUMBER OF STIMULI IN INPUT STREAM    = ',INFO(2,1)
      WRITE(5,*) '    NUMBER OF CHARACTERS IN INPUT STREAM = ',JNCNT
C ----- NCNT
      NCNT = INFO(2,1)
C -----   NELEM
      NELEM = 0
C -----   NGPM
      WRITE(5,*)
      WRITE(5,12)
      READ(5,*) NGPM
      IF((NGPM.LT.6).OR.(NGPM.GT.20)) THEN
           WRITE(5,*) ' GROUP SIZE INCORRECT; STOP PROGRAM '
           STOP
      END IF
C ----- RCRIT
      WRITE(5,*)
```

```
      WRITE(5,16)
      READ(5,*) RCRIT
C ----- KCOPB
      WRITE(5,*)
      WRITE(5,18)
      READ(5,*) KCOPB
C ----- NOUT
      WRITE(5,*)
      WRITE(5,14)
      READ(5,*) NOUT
C ----- NGROUP
      NGROUP = 5
C ----- NREP
      NREP = 50
C ----- NSTIM
      NSTIM = 2*NGROUP*NREP
C     CHECK THAT THERE ARE NOT TOO MANY STIMULI FOR PROGRAM TO HANDLE
      IF(INFO(2,1).GT.NSTIM) THEN
           WRITE(5,*) ' NUMBER OF ARRIVING STIMULI TOO LARGE: '
           WRITE(5,*) '   INFO(2,1) GT NSTIM = NREP*NGROUP '
           STOP
      END IF
C ----- NCHR
      NCHR = 31
C ----- NCHRBUF
      NCHRBUF = 410
C     CHECK THAT THERE ARE NOT TOO MANY STIMULI THAT MUST BE CONSTRUCTED
      IF(NGROUP*NREP.GT.NCHRBUF) THEN
           WRITE(5,*) ' NEED TO ENLARGE MATRICES CHRBUF AND STIMBUF '
           WRITE(5,*) '   SO THAT NGROUP*NREP GT NCHRBUF WHERE CHRBUF '
           WRITE(5,*) '   HAS DIMENSION NCHRBUF AND STIMBUF HAS DIMEN- '
           WRITE(5,*) '   SION 2*NCHRBUF '
           STOP
      END IF
C ----- NPER
      NPER = 0
      RETURN
      END
```

SUBROUTINE SINP

```
C
C
C
C
C
      SUBROUTINE SINP(STATEC,STATED,INFO,STIMDUR,ALPHA,BETA,
     C                  STATEX,STATEY)
      INTEGER INFO(2,4)
      INTEGER STATEC(30,4),STATEX(30,4)
      REAL ALPHA(2,4),BETA(2,4)
      REAL STATED(30,4),STATEY(30,4)
      REAL STIMDUR(43)
10    FORMAT(1X,4I4,10F10.4)
12    FORMAT(20(1X,12F5.0/))
14    FORMAT(8(1X,F7.2))
C
C     BEGIN INPUTTING INFORMATION
      DO 100 IA = 1,2
      READ(1,*) (INFO(IA,J),J=1,4)
100   CONTINUE
      DO 240 IA = 1,2
      READ(1,*) (ALPHA(IA,J),J=1,4),(BETA(IA,J),J=1,4)
240   CONTINUE
      DO 110 IA = 1,30
      DO 110 IB = 1,4
      STATEC(IA,IB) = 0.
      STATED(IA,IB) = 0.
110   CONTINUE
      STATEC(1,1) = 42              !INITIALIZE 0TH STIMULUS TO 42
      STATED(1,1) = 50.             !WITH A DURATION OF 50 MS
      DO 112 IA = 1,30
      DO 112 IB = 1,4
      STATEX(IA,IB) = STATEC(IA,IB)
      STATEY(IA,IB) = STATED(IA,IB)
112   CONTINUE
      DO 120 IA = 1,42,3
      READ(1,*) (STIMDUR(J),J=IA,IA+2)
120   CONTINUE
      READ(1,*) STIMDUR(43)
C
C     OUTPUT INFORMATION TO FILE ON WHAT WAS READ IN
      WRITE(5,*)
      WRITE(5,*) ' WARNING - WARNING : STIM DURATION OF SPEC CHAR WRONG '
      WRITE(5,*)
      WRITE(2,*) ' STIMULUS DURATIONS OF 1 - 31 CHARACTERS AND BLANKS '
      WRITE(2,12) (STIMDUR(J),J=1,33)
      WRITE(2,*)
      WRITE(2,*) ' DATA IN ALPHA AND BETA AT START '
      DO 260 IA = 1,2
      WRITE(2,14) (ALPHA(IA,J),J=1,4),(BETA(IA,J),J=1,4)
260   CONTINUE
      WRITE(2,*)
      WRITE(2,*) '    PERCEPTION (SERV/DEC): ',ALPHA(1,2)*BETA(1,2),
```

```
     C                                   ALPHA(2,2)*BETA(2,2)
     WRITE(2,*) '    PERCEPTION (SERV/DEC): ',ALPHA(1,3)*BETA(1,3),
     C                                   ALPHA(2,3)*BETA(2,3)
     WRITE(2,*) '    PERCEPTION (SERV/DEC): ',ALPHA(1,4)*BETA(1,4),
     C                                   ALPHA(2,4)*BETA(2,4)
     WRITE(2,*)
     WRITE(2,*) ' DATA IN INFO AT START '
     DO 220 IA = 1,2
     WRITE(2,*) (INFO(IA,J),J=1,4)
220  CONTINUE
     WRITE(2,*)
     WRITE(2,*) ' INPUT IN STATEC AND STATED AT START '
     DO 200 IA = 1,5
     WRITE(2,10) (STATEC(IA,J),J=1,4),(STATED(IA,J),J=1,4)
200  CONTINUE
     WRITE(2,*)
     WRITE(2,*) ' INPUT IN STATEX AND STATEY AT START '
     DO 210 IA = 1,5
     WRITE(2,10) (STATEX(IA,J),J=1,4),(STATEY(IA,J),J=1,4)
210  CONTINUE
C
C    CHECK ON VALIDITY OF INPUT INFORMATION
     DO 400 IA = 1,2
     DO 400 IB = 1,4
     IF(ALPHA(IA,IB).LE.1) THEN
          WRITE(5,*) ' ALPHA LESS THAN 1; ALTERNATIVE GAMMA GENERATOR '
               WRITE(5,*) '   IS NEEDED; SEE LAW AND KELTON, PAGE 255 '
          STOP
     END IF
400  CONTINUE
     RETURN
     END
```

SUBROUTINE SDAT

```
C
C
C
C
C
      SUBROUTINE SDAT(DATERR,DATLAT,CONMAT)
      REAL CONMAT(41,45)
      REAL DATERR(41,45),DATLAT(41,45)
10    FORMAT(1X,10F10.4)
      DO 100 IA = 1,41
      READ(3,10) (DATERR(IA,J),J=1,45)
100   CONTINUE
      DO 200 IA = 1,41
      READ(3,10) (DATLAT(IA,J),J=1,45)
200   CONTINUE
      DO 300 IA = 1,41
      RDEN = DATERR(IA,42) + DATERR(IA,43)
      IF(RDEN.GT.0) THEN
           IF(IA.EQ.1) THEN
                CONMAT(IA,1) = DATERR(IA,42)/RDEN
                CONMAT(IA,2) = 1.0
           ELSE
                CONMAT(IA,1) = DATERR(IA,43)/RDEN
                CONMAT(IA,IA) = 1.0
           END IF
      END IF
300   CONTINUE
      RETURN
      END
```

SUBROUTINE SCHRBUF

```
C
C
C
C
C
      SUBROUTINE SCHRBUF(SEED,CHRBUF,NCHR,NOUT)
      INTEGER NOUT
      INTEGER NCHR
      INTEGER*4 SEED
      INTEGER CHRBUF(410)
      INTEGER ICHR(41)
      REAL RCHR(41)
10    FORMAT(2(1X,16I4,/))
12    FORMAT(3(1X,12F6.4,/))
      DO 900 IX = 1,10
      DO 100 IA = 1,NCHR
      RCHR(IA) = RAN(SEED)
100   CONTINUE
      DO 200 IA = 1,NCHR
      RMIN = 2.
      DO 210 IB = 1,NCHR
      IF(RCHR(IB).LT.RMIN) THEN
           JPOS = IB
           RMIN = RCHR(IB)
      END IF
210   CONTINUE
      ICHR(JPOS) = IA
      RCHR(JPOS) = 2.
200   CONTINUE
      DO 300 IA = 1,NCHR
      JA = NCHR*(IX-1) + IA
      CHRBUF(JA) = ICHR(IA)
300   CONTINUE
900   CONTINUE
      RETURN
      END
```

SUBROUTINE SOUT

```
C
C
C
C
C
      SUBROUTINE SOUT(INFO,KEEP,KEEPSP,KILL,CONFUS,TIME,TIMESP,NCHR,
      C                    NCNT,NOUT,NGPM,NPER,DATERR,DATLAT,
      C                     RCRIT,KCOPB)
      INTEGER KCOPB
      INTEGER NCHR,NCNT,NOUT,NGPM,NPER
      INTEGER INFO(2,4)
      INTEGER CONFUS(41,41)
      INTEGER KEEP(41,46)
      INTEGER KEEPSP(5)
      INTEGER KILL(41,8)
      REAL RCRIT
      REAL DATERR(41,45),DATLAT(41,45)
      REAL DATOUT(2,8)
      REAL PRDERR(41,45)
      REAL PRDOUT(2,8)
      REAL TIME(41,45)
      REAL TIMESP(5)
10    FORMAT(1X,10I4)
12    FORMAT(15(1X,I4))
14    FORMAT(1X,I3,1X,F7.4,1X,100I3)
16    FORMAT(1X,I3,1X,15F7.1/5X,15F7.1/5X,15F7.1)
18    FORMAT(1X,10F12.6)
20    FORMAT(1X,10F12.4)
C     INITIALIZE MATRICES, VARIABLES
      DO 40 IA = 1,2
      DO 40 IB = 1,8
      PRDOUT(IA,IB) = 0.
      DATOUT(IA,IB) = 0.
40    CONTINUE
      DO 42 IA = 1,41
      DO 42 IB = 1,45
      PRDERR(IA,IB) = 0.
42    CONTINUE
      JKEEP = 0
      JKILL = 0
      JOFF = 0
      JPER = 0
      KPD = 0
      KPB = 0
      DO 80 IA = 1,NCHR
      DO 80 IB = 1,45
      IF(IB.EQ.44) THEN
           JPER = KEEP(IA,44) + JPER
      END IF
      IF(IB.NE.IA) THEN
           JOFF = KEEP(IA,IB) + JOFF
      END IF
      JKEEP = KEEP(IA,IB) + JKEEP
```

```
80    CONTINUE
      DO 85 IA = 1,NCHR
      DO 85 IB = 1,8
      IF(IB.EQ.2) KPB = KPB + KILL(IA,IB)
      IF(IB.EQ.6) KPD = KPD + KILL(IA,IB)
      JKILL = KILL(IA,IB) + JKILL
85    CONTINUE
      JNCNT = (NCNT - 1)/2 + 1
      RNCNT = JNCNT
      JKEEPS = 0
      DO 86 IA = 1,5
      JKEEPS = KEEPSP(IA) + JKEEPS
86    CONTINUE
C
C     FIND PREDICTED ERROR PERCENTAGES: NEED TO ASSIGN COUNTS TO
C          KEEP(IA,42) = KEEP(IA,IA), KEEP(IA,43) = CHARACTERS
C          TYPED AS JA.NE.IE, KEEP(IA,44) = PERIODS,
C          KEEP(IA,45) = NO RESPONSES, AND KEEP(IA,46) = KEEP(IA,1)
C          + ... + KEEP(IA,45).  USED AS DENOMINATOR FOR ROW
C          ERROR PROBABILITIES IN PRDERR
C     TO BEGIN, CONSTRUCT KEEP(IA,42)
      DO 400 IA = 1,41
      KEEP(IA,42) = KEEP(IA,IA)
C     NEXT CONSTRUCT KEEP(IA,43)
      JA = 0
      DO 403 IB = 1,41
      IF(IA.NE.IB) THEN
           JA = KEEP(IA,IB) + JA
      END IF
403   CONTINUE
      KEEP(IA,43) = JA
C     KEEP(IA,44) ALREADY CONSTRUCTED SINCE RUNNING COUNT KEPT OF PERIODS
C     CONSTRUCT KEEP(IA,45)
      DO 404 IB = 1,8
      KEEP(IA,45) = KILL(IA,IB) + KEEP(IA,45)
404   CONTINUE
C     KEEP(IA,46) SET TO SUM OF ALL KEEP + KILL RESPONSES TO STIMULUS IA.
C          AT THE SAME TIME COMPUTE KEEP TO ERROR PERCENTAGES.
      DO 406 IB = 1,45
      IF(KEEP(IA,46).NE.0) THEN
           RNUM = KEEP(IA,IB)
           RDEN = KEEP(IA,46)
           PRDERR(IA,IB) = RNUM/RDEN
      END IF
406   CONTINUE
400   CONTINUE
C     COMPUTE AVERAGE ERROR PERCENTAGES FOR CHARACTER CORRECT,
C          CHARACTER INCORRECT, PERIOD AND NO RESPONSE
      DO 410 IA = 1,41
      DO 412 IB = 42,45
      PRDOUT(1,IB-41) = KEEP(IA,IB) + PRDOUT(1,IB-41)
412   CONTINUE
410   CONTINUE
      DO 414 IA = 1,4
```

```
      PRDOUT(2,IA) = PRDOUT(1,IA)/RNCNT
414   CONTINUE
C     ALSO NEED TO FIND OBSERVED SUMMARY ERROR PERCENTAGES
      DO 420 IA = 1,41
      RSUM = 0.
      DO 422 IB = 42,45
      RSUM = DATERR(IA,IB) + RSUM
422   CONTINUE
      DO 424 IB = 42,45
      IF(RSUM.GT.0) THEN
           DATOUT(1,IB-41) = DATOUT(1,IB-41) + 1.
           DATOUT(2,IB-41) = DATOUT(2,IB-41) + DATERR(IA,IB)
      END IF
424   CONTINUE
420   CONTINUE
      DO 426 IA = 1,4
      IF(DATOUT(1,IA).NE.0) THEN
           DATOUT(2,IA) = DATOUT(2,IA)/DATOUT(1,IA)
      END IF
426   CONTINUE
C
C     ALSO NEED TO COMPUTE RAW SUMS FOR PREDICTED LATENCIES; RAW SUMS
C          ALREADY RECORDED FOR TIME(IA,J), J = 1,...,41.  TIME(IA,42)
C          SET TO TIME(IA,IA); TIME(IA,43) SET TO SUM OF CHARACTER
C          ONLY RESPONSE TIMES; AND TIME(IA,44) ALREADY SET.  NOTE
C          THERE IS NO ENTRY IN TIME(IA,45) SINCE THERE IS NO TIME
C          FOR NO RESPONSES.
      DO 440 IA = 1,41
      TIME(IA,42) = TIME(IA,IA)
      DO 440 IB = 1,41
      IF(IA.NE.IB) THEN
           TIME(IA,43) = TIME(IA,IB) + TIME(IA,43)
      END IF
440   CONTINUE
C     FIND WEIGHTED AVERAGE SIMULATED (PREDICTED) LATENCIES.
      DO 450 IA = 1,41
      DO 450 IB = 42,45
           PRDOUT(1,IB-37) = PRDOUT(1,IB-37) + TIME(IA,IB)
450   CONTINUE
      DO 452 IA = 5,8
      IF(PRDOUT(1,IA-4).NE.0) THEN
           PRDOUT(2,IA) = PRDOUT(1,IA)/PRDOUT(1,IA-4)
      END IF
452   CONTINUE
C     COMPUTE TIMES IN EACH CELL; CANNOT DO BEFORE ABOVE BECAUSE
C          NEED TO KEEP SUMS IN TIME IN ORDER TO GET CORRECT
C          WEIGHTED AVERAGES.
      DO 454 IA = 1,NCHR
      DO 454 IB = 1,NCHR
      IF(KEEP(IA,IB).NE.0) THEN
           RA = KEEP(IA,IB)
           TIME(IA,IB) = TIME(IA,IB)/RA
      END IF
454   CONTINUE
```

```
C     FIND AVERAGE OBSERVED LATENCIES
      DO 460 IA = 1,41
      DO 464 IB = 42,45
      IF(DATLAT(IA,IB).GT.0) THEN
           DATOUT(1,IB-37) = DATOUT(1,IB-37) + 1.
           DATOUT(2,IB-37) = DATOUT(2,IB-37) + DATLAT(IA,IB)
      END IF
      IF(IB.EQ.42) THEN
      END IF
464   CONTINUE
460   CONTINUE
      DO 466 IB = 5,8
      IF(DATOUT(1,IB).NE.0) THEN
           DATOUT(2,IB) = DATOUT(2,IB)/DATOUT(1,IB)
      END IF
466   CONTINUE
C
C     WRITE OUT SUMMARY INFORMATION
      RKEEPS = JKEEPS
      RPC = RKEEPS/RNCNT
      WRITE(2,*)
      WRITE(2,*) ' COPY BEHIND = 1, NO COPY BEHIND = 0 ',KCOPB
      WRITE(2,*)
      WRITE(2,*) ' GROUPS PER MINUTE (GPM)          = ',NGPM
      WRITE(2,*)
      WRITE(2,*) ' CRITICAL TIME AT PERC SERVER     = ',RCRIT
      WRITE(2,*)
      WRITE(2,*) ' TOT CHAR, KEEP, KILL             = ',JNCNT,JKEEP,JKILL
      WRITE(2,*)
      WRITE(2,*) ' TOT CHAR, TOT COR, % COR         = ',JNCNT,JKEEPS,RPC
      WRITE(2,*)
      WRITE(2,*) ' P(CORRECT), P(BAD CHAR), P(PERIOD), P(NO RESP) '
      WRITE(2,18) (PRDOUT(2,J),J=1,4)
      WRITE(2,18) (DATOUT(2,J),J=1,4)
      WRITE(2,*)
      WRITE(2,*) ' RT(CORREC), RT(BAD CHR), RT(PERIO), RT(NO RSP) '
      WRITE(2,20) (PRDOUT(2,J),J=5,8)
      WRITE(2,20) (DATOUT(2,J),J=5,8)
      WRITE(2,*)
      WRITE(2,*) ' SERIAL POSITION INFORMATION: CORRECT KEEP '
      WRITE(2,*) (KEEPSP(J),J=1,5)
      DO 88 IA = 1,5
      RA = KEEPSP(IA)
      IF(KEEPSP(IA).NE.0) TIMESP(IA) = TIMESP(IA)/RA
88    CONTINUE
      WRITE(2,*)
      WRITE(2,*) ' SERIAL POSITION INFORMATION: CORRECT TIME '
      WRITE(2,*) (TIMESP(J),J=1,5)
C
C     WRITE OUT DETAILED INFORMATION
      IF(NOUT.EQ.0) GO TO 999
      WRITE(2,*)
      WRITE(2,*) ' KEEP INFORMATION '
      DO 90 IA = 1,NCHR
```

```
      WRITE(2,12) (KEEP(IA,J),J=1,45)
90    CONTINUE
      WRITE(2,*)
      WRITE(2,*) ' KILL INFORMATION '
      DO 100 IA = 1,NCHR
      WRITE(2,10) IA,(KILL(IA,J),J=1,8)
100   CONTINUE
      WRITE(2,*)
      WRITE(2,*) ' CONFUSION INFORMATION FROM SIMULATION '
      DO 105 IA = 1,NCHR
      RSUM = 0.
      DO 110 IB = 1,NCHR
      RSUM = CONFUS(IA,IB) + RSUM
110   CONTINUE
      RNUM = CONFUS(IA,IA)
      IF(RSUM.NE.0) THEN
           RPER = RNUM/RSUM
           WRITE(2,14) IA,RPER,(CONFUS(IA,J),J=1,NCHR)
      ELSE
           WRITE(2,*) IA,' RSUM = 0: NO CHARACTERS SENT OR ALL BUMPED '
      END IF
105   CONTINUE
      WRITE(2,*)
      WRITE(2,*) ' RESPONSE TIME INFORMATION '
      DO 210 IA = 1,NCHR
      WRITE(2,16) IA,(TIME(IA,J),J=1,NCHR)
210   CONTINUE
999   CONTINUE
C
C     PERFORM CHECKS ON COMPUTATIONS
C          CHECK 1
      NTOT = 0
      DO 300 IA = 1,NCHR
      DO 300 IB = 1,NCHR
      NTOT = CONFUS(IA,IB) + NTOT
300   CONTINUE
      IF(NTOT.NE.(JKEEP + JKILL - KPD - KPB - NPER)) THEN
           WRITE(2,*)
           WRITE(2,*) '   CHECK 1 PROBLEMS: CONFUS SUM NE KEEP + KILL - '
           WRITE(2,*) '   KILLPERBUFDECAY - KILLPERBUFBUMP - # REC PER'
           WRITE(2,*) '  ',NTOT,JKEEP,JKILL,KPD,KPB,NPER
           WRITE(5,*)
           WRITE(5,*) '   CHECK 1 PROBLEMS: CONFUS SUM NE KEEP + KILL - '
           WRITE(5,*) '   KILLPERBUFDECAY - KILLPERBUFBUMP - # REC PER'
           WRITE(5,*) '  ',NTOT,JKEEP,JKILL,KPD,KPB,NPER
      ELSE
           WRITE(5,*)
           WRITE(5,*) '   CHECK 1 OK: '
           WRITE(5,*) '   CONFUS = KEEP + KILL - KPD - KPB - NPER'
           WRITE(5,*) '  ',NTOT,JKEEP,JKILL,KPD,KPB,NPER
      END IF
C     PERFORM CHECKS ON COMPUTATIONS
C          CHECK 2
      IF(JNCNT.NE.(JKEEP+JKILL)) THEN
```

```
            WRITE(2,*)
            WRITE(2,*) '   CHECK 2 PROBLEMS: NUMBER OF STIMULI IN STREAM '
            WRITE(2,*) '   NOT EQUAL TO KEEP + KILL COUNT '
            WRITE(2,*) '   ',JNCNT,JKEEP,JKILL
            WRITE(5,*)
            WRITE(5,*) '   CHECK 2 PROBLEMS: NUMBER OF STIMULI IN STREAM '
            WRITE(5,*) '   NOT EQUAL TO KEEP + KILL COUNT '
            WRITE(5,*) '   ',JNCNT,JKEEP,JKILL
      ELSE
            WRITE(5,*)
            WRITE(5,*) '   CHECK 2 OK: KEEP + KILL = # CHAR IN INPUT '
            WRITE(5,*) '   ',JNCNT,JKEEP,JKILL
      END IF
C     PERFORM CHECK ON COMPUTATIONS
C     CHECK 3
      IF(JNCNT.NE.(JKEEPS+JKILL+JOFF)) THEN
            WRITE(2,*)
            WRITE(2,*) '   CHECK 3 PROBLEMS: NUMBER OF STIMULI IN STREAM '
            WRITE(2,*) '   NOT EQUAL TO CORRECT + KILL + OFF DIAG KEEP '
            WRITE(2,*) '  ',JNCNT,JKEEPS,JKILL,JOFF
            WRITE(5,*)
            WRITE(5,*) '   CHECK 3 PROBLEMS: NUMBER OF STIMULI IN STREAM '
            WRITE(5,*) '   NOT EQUAL TO CORRECT + KILL + OFF DIAG KEEP '
            WRITE(5,*) '  ',JNCNT,JKEEPS,JKILL,JOFF
      ELSE
            WRITE(5,*)
            WRITE(5,*) '   CHECK 3 OK: NUMBER OF STIMULI IN STREAM '
            WRITE(5,*) '   IS EQUAL TO CORRECT + KILL + OFF DIAG KEEP '
            WRITE(5,*) '  ',JNCNT,JKEEPS,JKILL,JOFF
      END IF
      RETURN
      END
```

SUBROUTINE SSTIM

```
C
C
C
C
C
      SUBROUTINE SSTIM(CHRBUF,STIMBUF,NGROUP,NREP,NSTIM,NOUT)
      INTEGER CHRBUF(410),STIMBUF(820)
      INTEGER NGROUP,NREP,NSTIM
10    FORMAT(1X,20I3)
      JCHR = 0
      DO 100 IA = 1,NSTIM,2*NGROUP
      DO 120 IB = IA,IA+2*NGROUP-1,2
      JCHR = JCHR + 1
      STIMBUF(IB) = CHRBUF(JCHR)
      STIMBUF(IB+1) = 42
120   CONTINUE
      STIMBUF(IA + 2*NGROUP -1) = 43
100   CONTINUE
      IF(NOUT.EQ.1) THEN
           WRITE(2,*)
           WRITE(2,*) ' DATA IN CHRBUF AT START: ONLY FIRST 10 '
           WRITE(2,10) (CHRBUF(J),J=1,10)
           WRITE(2,*)
           WRITE(2,*) ' DATA IN STIMBUF AT START: ONLY FIRST 20 '
           WRITE(2,10) (STIMBUF(J),J=1,20)
      END IF
      RETURN
      END
```

SUBROUTINE SMINI

```
C
C
C
C
C
      SUBROUTINE SMINI(STATEC,STATED,INFO,KROW,KCOL,NOUT)
      INTEGER NOUT
      INTEGER INFO(2,4)
      INTEGER STATEC(30,4)
      REAL STATED(30,4)
      INTEGER KROW,KCOL
      RMIN = 10000000.
      DO 100 IA = 1,4
      IF(INFO(1,IA).GT.0) THEN
           DO 120 IB = 1,INFO(1,IA)
           IF(RMIN.GT.STATED(IB,IA)) THEN
                KROW = IB
                KCOL = IA
                RMIN = STATED(KROW,KCOL)
           END IF
120        CONTINUE
      END IF
100   CONTINUE
      IF(NOUT.EQ.1) THEN
           WRITE(2,*) ' MIN DURATION IN STATE ',KROW,KCOL,RMIN
      END IF
      RETURN
      END
```

SUBROUTINE SSUB

```
C
C
C
C
C
      SUBROUTINE SSUB(STATEC,STATED,INFO,KROW,KCOL)
      INTEGER INFO(2,4)
      INTEGER STATEC(30,4)
      REAL STATED(30,4)
      INTEGER KROW,KCOL
10    FORMAT(1X,4I4,10F10.4)
      RMIN = STATED(KROW,KCOL)                    !THE MINIMUM FOUND IN SMIN
      DO 100 IA = 1,4
      IF(INFO(1,IA).EQ.0) GOTO 100
      DO 100 IB = 1,INFO(1,IA)
      STATED(IB,IA) = STATED(IB,IA) - RMIN
100   CONTINUE
      RETURN
      END
```

```
C
C
C
C
C
      SUBROUTINE SARR(STATEC,STATED,INFO,KROW,KCOL,STIMBUF,NELEM,
     C              KILL,SEED,STIMDUR,ALPHA,BETA,STATEX,STATEY,STATEP,
     C              NOUT,DATLAT,KCOPB)
      INTEGER KROW,KCOL
      INTEGER NELEM
      INTEGER NOUT
      INTEGER*4 SEED
      INTEGER INFO(2,4)
      INTEGER STATEC(30,4),STATEX(30,4)
      INTEGER STATEP(30,4)
      INTEGER STIMBUF(820)
      INTEGER KILL(41,8)
      REAL ALPHA(2,4),BETA(2,4)
      REAL DATLAT(41,45)
      REAL STATED(30,4),STATEY(30,4)
      REAL STIMDUR(43)
10    FORMAT(1X,4I4,10F10.4)
C     MAKE SURE THAT SARR IS CALLED WHEN IT WAS POSSIBLE FOR ARRIVAL TO
C          HAVE HAD MINIMUM DURATION
      IF(INFO(1,1).EQ.0) THEN
           WRITE(2,*) ' NOT POSSIBLE FOR MINIMUM TO HAVE BEEN ARRIVAL '
      END IF
C     CHANGE ARRIVAL STATES
      JTEMPC = STATEC(1,1)
      JTEMPP = STATEP(1,1)
      IF(INFO(2,1).EQ.0) THEN                !NO MORE ARRIVALS
           STATEC(1,1) = 0
           STATED(1,1) = 0.
           STATEP(1,1) = 0
           STATEX(1,1) = 0
           STATEY(1,1) = 0.
      END IF
      IF(INFO(2,1).GT.0) THEN                !MORE ARRIVALS
           STATEC(1,1) = STIMBUF(NELEM)
           STATED(1,1) = STIMDUR(STIMBUF(NELEM))
           IF(STATEC(1,1).EQ.43) THEN
                STATEP(1,1) = 0
           END IF
           IF(STATEC(1,1).LT.42) THEN
                STATEP(1,1) = STATEP(1,1) + 1
           END IF
           STATEX(1,1) = STATEC(1,1)
           STATEY(1,1) = 0.
      END IF
C     CHANGE ARRIVAL INFORMATION
      IF(INFO(2,1).EQ.0) THEN
           INFO(1,1) = 0
      END IF
```

```
      IF(INFO(2,1).GT.0) THEN
           INFO(2,1) = INFO(2,1) - 1
      END IF
C     CHANGE PERCEPTION STATES
      IF(JTEMPC.GE.42) THEN                    !NO CHANGE NEEDED IF CHAR ON
C                                                                    ARRIVAL
SERVER IS BLANK
           GOTO 999
      END IF
C
C     CODE BELOW USED TO MODEL BEHAVIOR OF OPERATORS WHO CAN WORK ON
C          AT MOST ONE STIMULUS AT A TIME, I.E., OF OEPRATORS WHO
C          CANNOT COPY BEHIND
      IF(KCOPB.EQ.0) THEN
           RSTAT = INFO(1,2) + INFO(1,3) + INFO(1,4)
           IF(RSTAT.GT.0) THEN              !AT LEAST 1 SERVER OCCUPIED
                KILL(JTEMPC,2) = KILL(JTEMPC,2) + 1
                IF(NOUT.EQ.1) THEN
                                                 WRITE(2,*)
                                                 WRITE(2,*) ' KILLED
CHARACTER: PER BUFF FULL ',
      C
      JTEMPC,KILL(JTEMPC,2)
                END IF
                GOTO 999
           END IF
      END IF
      IF(INFO(1,2).EQ.0) THEN                  !ROOM ON PER SERVER
           STATEC(1,2) = JTEMPC
           STATEX(1,2) = JTEMPC
           STATEP(1,2) = JTEMPP
           STATED(1,2) = SGAMMA(ALPHA,BETA,DATLAT,SEED,
      C                              1,2,1,JTEMPC)
           STATEY(1,2) = STATED(1,2)
      END IF
      IF((INFO(1,2).GT.0).AND.(INFO(1,2).LT.INFO(2,2))) THEN
C                                              !MORE ROOM IN PER BUFFER
           STATEC(INFO(1,2)+1,2) = JTEMPC
           STATEX(INFO(1,2)+1,2) = JTEMPC
           STATEP(INFO(1,2)+1,2) = JTEMPP
           STATED(INFO(1,2)+1,2) = SGAMMA(ALPHA,BETA,DATLAT,
      C                                   SEED,2,2,1,JTEMPC)
           STATEY(INFO(1,2)+1,2) = STATED(1,2)
      END IF
      IF(INFO(1,2).GE.INFO(2,2)) THEN          !NO MORE ROOM IN BUFFER
           KILL(JTEMPC,2) = KILL(JTEMPC,2) + 1
           IF(NOUT.EQ.1) THEN
                WRITE(2,*)
                WRITE(2,*) ' KILLED CHARACTER: PER BUFF FULL ',
      C                                         JTEMPC,KILL(JTEMPC,2)
           END IF
      END IF
C     CHANGE PERCEPTION INFORMATION
      IF(INFO(1,2).LT.INFO(2,2)) THEN
```

```
          INFO(1,2) = INFO(1,2) + 1
      END IF
999   CONTINUE
      IF(NOUT.EQ.1) THEN
          WRITE(2,*)
          WRITE(2,*) ' INPUT AFTER SARR CHANGES '
          DO 910 IA = 1,2
          WRITE(2,10) (INFO(IA,J),J=1,4)
910       CONTINUE
          DO 900 IA = 1,2
          WRITE(2,10) (STATEC(IA,J),J=1,4),(STATED(IA,J),J=1,4)
900       CONTINUE
          DO 902 IA = 1,2
          WRITE(2,10) (STATEX(IA,J),J=1,4),(STATEY(IA,J),J=1,4)
902       CONTINUE
          DO 904 IA = 1,2
          WRITE(2,10) (STATEP(IA,J),J=1,4)
904       CONTINUE
      END IF
      RETURN
      END
```

```
C
C
C
C
C
      SUBROUTINE SPER(STATEC,STATED,INFO,KROW,KCOL,STIMBUF,NELEM,
      C          KILL,CONMAT,CONFUS,SEED,ALPHA,BETA,STATEX,STATEY,STATEP,
      C          NOUT,NPER,RCRIT,DATLAT)
      INTEGER KROW,KCOL
      INTEGER NELEM
      INTEGER NOUT
      INTEGER NPER
      INTEGER*4 SEED
      INTEGER CONFUS(41,41)
      INTEGER INFO(2,4)
      INTEGER STATEC(30,4),STATEX(30,4)
      INTEGER STATEP(30,4)
      INTEGER STIMBUF(820)
      INTEGER KILL(41,8)
      REAL RCRIT
      REAL ALPHA(2,4),BETA(2,4)
      REAL DATLAT(41,45)
      REAL STATED(30,4),STATEY(30,4)
      REAL CONMAT(41,45)
10    FORMAT(1X,4I4,10F10.4)
C     MAKE SURE THAT SPER IS CALLED WHEN IT WAS POSSIBLE FOR PERCEPTION TO
C          HAVE HAD MINIMUM DURATION
      IF(INFO(1,2).EQ.0) THEN
           WRITE(2,*) ' NOT POSSIBLE FOR MINIMUM TO HAVE BEEN PERCEPTION '
      END IF
C     INITIALIZE VARIABLES
      JTEMPC = STATEC(1,2)
      JTEMPX = STATEX(1,2)
      JTEMPP = STATEP(1,2)
      RTEMPY = STATEY(1,2)
C     IF THE PERCEPTION TIME IS LONG, THEN A PERIOD WILL BE RECORDED
      IF(RTEMPY.GT.RCRIT) THEN
           JTEMPC = 44
           NPER = NPER + 1
      END IF
C     IF PERCEPTION TIME IS LESS THAN RCRIT, THEN GOTO CONFUSION MATRIX
C          TO CREATE PERCEPTUAL ERRORS, E.G., IF THE CODE -...
C          IS ON THE PERCEPTION SERVER, THEN WTIH SOME PROBABILITY
C          THE CODE MIGHT BE PERCEIVED AS, SAY, -..
      IF(RTEMPY.LE.RCRIT) THEN
           CALL SPERERR(JTEMPC,CONMAT,CONFUS,SEED)
      END IF
C     CHANGE PERCEPTION STATES
      IF(INFO(1,2).EQ.1) THEN                !NOTHING IN PERCEPTION BUFFER
           STATEC(1,2) = 0
           STATED(1,2) = 0.
           STATEX(1,2) = 0
           STATEP(1,2) = 0
```

```
          STATEY(1,2) = 0.
     END IF
     IF(INFO(1,2).GT.1) THEN                                    !MORE IN
PERCEPTION BUFFER
          STATEC(1,2) = STATEC(2,2)      !ASSIGN DURATION PERC SERVER
          STATED(1,2) = SGAMMA(ALPHA,BETA,DATLAT,SEED,
     C                                 1,2,2,JTEMPX)
          STATEX(1,2) = STATEX(2,2)
          STATEP(1,2) = STATEP(2,2)
          STATEY(1,2) = STATEY(2,2) + STATED(1,2)
          DO 100 IA = 2,INFO(1,2)
          STATEC(IA,2) = STATEC(IA+1,2)
          STATED(IA,2) = STATED(IA+1,2)
          STATEX(IA,2) = STATEX(IA+1,2)
          STATEP(IA,2) = STATEP(IA+1,2)
          IF(IA.LT.INFO(1,2)) THEN
               STATEY(IA,2) = STATEY(IA+1,2) + STATED(1,2)
          ELSE
               STATEY(IA,2) = 0.
          END IF
100       CONTINUE
     END IF
C    CHANGE PERCEPTION INFORMATION
     INFO(1,2) = INFO(1,2) -1
C    CHANGE RECOGNITION STATES
     IF(INFO(1,3).LT.INFO(2,3)) THEN          !MORE ROOM IN RECOG BUFFER
          STATEC(INFO(1,3)+1,3) = JTEMPC  !ASSIGN DURATIONS
          STATEX(INFO(1,3)+1,3) = JTEMPX
          STATEP(INFO(1,3)+1,3) = JTEMPP
          IF(INFO(1,3).EQ.0) THEN
          STATED(1,3) = SGAMMA(ALPHA,BETA,DATLAT,SEED,
     C                                 1,3,2,JTEMPX)
          ELSE
          STATED(INFO(1,3)+1,3) = SGAMMA(ALPHA,BETA,DATLAT,
     C                                      SEED,2,3,2,JTEMPX)
          END IF
          STATEY(INFO(1,3)+1,3) = RTEMPY + STATED(1,3)
     END IF
     IF(INFO(1,3).GE.INFO(2,3)) THEN          !NO MORE ROOM IN RECOG BUFFER
          KILL(JTEMPX,3) = KILL(JTEMPX,3) + 1
          IF(NOUT.EQ.1) THEN
               WRITE(2,*)
               WRITE(2,*) ' KILLED CHARACTER: REC BUFF FULL ',
     C                                         JTEMPX,KILL(JTEMPX,3)
          END IF
     END IF
C    CHANGE RECOGNITION INFORMATION
     IF(INFO(1,3).LT.INFO(2,3)) THEN
          INFO(1,3) = INFO(1,3) + 1
     END IF
999  CONTINUE
     IF(NOUT.EQ.1) THEN
          WRITE(2,*)
          WRITE(2,*) ' INPUT AFTER SPER CHANGES '
```

```
          DO 910 IA = 1,2
          WRITE(2,10) (INFO(IA,J),J=1,4)
910       CONTINUE
          DO 900 IA = 1,2
          WRITE(2,10) (STATEC(IA,J),J=1,4),(STATED(IA,J),J=1,4)
900       CONTINUE
          DO 902 IA = 1,2
          WRITE(2,10) (STATEX(IA,J),J=1,4),(STATEY(IA,J),J=1,4)
902       CONTINUE
          DO 904 IA = 1,2
          WRITE(2,10) (STATEP(IA,J),J=1,4)
904       CONTINUE
     END IF
     RETURN
     END
```

```
C
C
C
C
C
      SUBROUTINE SREC(STATEC,STATED,INFO,KROW,KCOL,STIMBUF,NELEM,
      C        KILL,SEED,ALPHA,BETA,STATEX,STATEY,STATEP,NOUT,
      C        DATLAT)
      INTEGER KROW,KCOL
      INTEGER NELEM
      INTEGER NOUT
      INTEGER*4 SEED
      INTEGER INFO(2,4)
      INTEGER STATEC(30,4),STATEX(30,4)
      INTEGER STATEP(30,4)
      INTEGER STIMBUF(820)
      INTEGER KILL(41,8)
      REAL ALPHA(2,4),BETA(2,4)
      REAL DATLAT(41,45)
      REAL STATED(30,4),STATEY(30,4)
10    FORMAT(1X,4I4,10F10.4)
C     MAKE SURE THAT SREC IS CALLED WHEN IT WAS POSSIBLE FOR PERCEPTION TO
C          HAVE HAD MINIMUM DURATION
      IF(INFO(1,3).EQ.0) THEN
           WRITE(2,*)
           WRITE(2,*) ' NOT POSSIBLE FOR MINIMUM TO HAVE BEEN RECOGNITION '
      END IF
C     CHANGE RECOGNITION STATES
      JTEMPC = STATEC(1,3)
      JTEMPX = STATEX(1,3)
      JTEMPP = STATEP(1,3)
      RTEMPY = STATEY(1,3)
      IF(INFO(1,3).EQ.1) THEN                !NOTHING IN RECOGNITION BUFFER
           STATEC(1,3) = 0
           STATED(1,3) = 0.
           STATEX(1,3) = 0
           STATEP(1,3) = 0
           STATEY(1,3) = 0.
      END IF
      IF(INFO(1,3).GT.1) THEN                                    !MORE IN
RECOGNITION BUFFER
           STATEC(1,3) = STATEC(2,3)       !ASSIGN DURATION TO RECOG SERVER
           STATED(1,3) = SGAMMA(ALPHA,BETA,DATLAT,SEED,
      C                                 1,3,3,JTEMPX)
           STATEX(1,3) = STATEX(2,3)
           STATEP(1,3) = STATEP(2,3)
           STATEY(1,3) = STATEY(2,3) + STATED(1,3)
           DO 100 IA = 2,INFO(1,3)
           STATEC(IA,3) = STATEC(IA+1,3)
           STATED(IA,3) = STATED(IA+1,3)
           STATEX(IA,3) = STATEX(IA+1,3)
           STATEP(IA,3) = STATEP(IA+1,3)
           IF(IA.LT.INFO(1,3)) THEN
```

```
            STATEY(IA,3) = STATEY(IA+1,3) + STATED(1,3)
          ELSE
            STATEY(IA,3) = 0.
          END IF
100       CONTINUE
      END IF
C     CHANGE RECOGNITION INFORMATION
      INFO(1,3) = INFO(1,3) -1
C     CHANGE EXECUTION STATES
      IF(INFO(1,4).EQ.0) THEN                    !ROOM ON EXEC SERVER
          STATEC(1,4) = JTEMPC
          STATEX(1,4) = JTEMPX
          STATEP(1,4) = JTEMPP
          STATED(1,4) = SGAMMA(ALPHA,BETA,DATLAT,SEED,
     C                              1,4,3,JTEMPX)
          STATEY(1,4) = RTEMPY + STATED(1,4)
      END IF
      IF((INFO(1,4).GT.0).AND.(INFO(1,4).LT.INFO(2,4))) THEN
C                                                !MORE ROOM IN EXEC BUFFER
          STATEC(INFO(1,4)+1,4) = JTEMPC
          STATEX(INFO(1,4)+1,4) = JTEMPX
          STATEP(INFO(1,4)+1,4) = JTEMPP
          STATED(INFO(1,4)+1,4) = SGAMMA(ALPHA,BETA,DATLAT,
     C                                   SEED,2,4,3,JTEMPX)
          STATEY(INFO(1,4)+1,4) = RTEMPY + STATED(1,4)
      END IF
      IF(INFO(1,4).GE.INFO(2,4)) THEN          !NO MORE ROOM IN EXEC BUFFER
          KILL(JTEMPX,4) = KILL(JTEMPX,4) + 1
          IF(NOUT.EQ.1) THEN
              WRITE(2,*)
              WRITE(2,*) ' KILLED CHARACTER: EXEC BUFF FULL ',
     C                                      JTEMPX,KILL(JTEMPX,4)
          END IF
      END IF
C     CHANGE EXECUTION INFORMATION
      IF(INFO(1,4).LT.INFO(2,4)) THEN
          INFO(1,4) = INFO(1,4) + 1
      END IF
999   CONTINUE
      IF(NOUT.EQ.1) THEN
          WRITE(2,*)
          WRITE(2,*) ' INPUT AFTER SREC CHANGES '
          DO 910 IA = 1,2
          WRITE(2,10) (INFO(IA,J),J=1,4)
910       CONTINUE
          DO 900 IA = 1,2
          WRITE(2,10) (STATEC(IA,J),J=1,4),(STATED(IA,J),J=1,4)
900       CONTINUE
          DO 902 IA = 1,2
          WRITE(2,10) (STATEX(IA,J),J=1,4),(STATEY(IA,J),J=1,4)
902       CONTINUE
          DO 904 IA = 1,2
          WRITE(2,10) (STATEP(IA,J),J=1,4)
904       CONTINUE
```

```
END IF
RETURN
END
```

SUBRUOTINE SEXE

```
C
C
C
C
C
      SUBROUTINE SEXE(STATEC,STATED,INFO,KROW,KCOL,STIMBUF,NELEM,
      C        KEEP,KILL,ALPHA,BETA,STATEX,STATEY,TIME,STATEP,
      C        KEEPSP,TIMESP,NOUT,DATLAT)
      INTEGER KROW,KCOL
      INTEGER NELEM
      INTEGER NOUT
      INTEGER INFO(2,4)
      INTEGER STATEC(30,4),STATEX(30,4)
      INTEGER STATEP(30,4)
      INTEGER STIMBUF(820)
      INTEGER KILL(41,8)
      INTEGER KEEP(41,46)
      INTEGER KEEPSP(5)
      REAL ALPHA(2,4),BETA(2,4)
      REAL DATLAT(41,45)
      REAL STATED(30,4),STATEY(30,4)
      REAL TIME(41,45)
      REAL TIMESP(5)
10    FORMAT(1X,4I4,10F10.4)
C     MAKE SURE THAT SEXE IS CALLED WHEN IT WAS POSSIBLE FOR PERCEPTION TO
C          HAVE HAD MINIMUM DURATION
      IF(INFO(1,4).EQ.0) THEN
           WRITE(2,*)
           WRITE(2,*) ' NOT POSSIBLE FOR MINIMUM TO HAVE BEEN EXECUTION '
      END IF
C
C     SAVE INFORMATION ON CHARACTERS WHICH MAKE IT THROUGH RESPONSE
C          AND INCREMENT RUNING RESPONSE TIMES
      KEEP(STATEX(1,4),STATEC(1,4)) = KEEP(STATEX(1,4),STATEC(1,4)) + 1
      TIME(STATEX(1,4),STATEC(1,4)) = TIME(STATEX(1,4),STATEC(1,4)) +
      C        STATEY(1,4)
C
C     SAVE SERIAL POSITION INFORMATION FOR CHARACTERS WHICH ARE CORRECTLY
C          ENTERED
      IF(STATEX(1,4).EQ.STATEC(1,4)) THEN
           KEEPSP(STATEP(1,4)) = KEEPSP(STATEP(1,4)) + 1
           TIMESP(STATEP(1,4)) = TIMESP(STATEP(1,4)) + STATEY(1,4)
      END IF
C     CHANGE EXECUTION STATES
      IF(INFO(1,4).EQ.1) THEN                    !NOTHING IN EXECUTION BUFFER
           STATEC(1,4) = 0
           STATED(1,4) = 0.
           STATEX(1,4) = 0
           STATEP(1,4) = 0
           STATEY(1,4) = 0.
      END IF
      IF(INFO(1,4).GT.1) THEN                                    !MORE IN
EXECUTION BUFFER
```

```
          STATEC(1,4) = STATEC(2,4)
          STATED(1,4) = SGAMMA(ALPHA,BETA,DATLAT,SEED,
     C                                 1,4,4,JTEMPX)
          STATEX(1,4) = STATEX(2,4)
          STATEP(1,4) = STATEP(2,4)
          STATEY(1,4) = STATEY(2,4) + STATED(1,4)
          DO 100 IA = 2,INFO(1,4)
          STATEC(IA,4) = STATEC(IA+1,4)
          STATED(IA,4) = STATED(IA+1,4)
          STATEX(IA,4) = STATEX(IA+1,4)
          STATEP(IA,4) = STATEP(IA+1,4)
          IF(IA.LT.INFO(1,4)) THEN
               STATEY(IA,4) = STATEY(IA+1,4) + STATED(1,4)
          ELSE
               STATEY(IA,4) = 0.
          END IF
100       CONTINUE
     END IF
C    CHANGE EXECUTION INFORMATION
     INFO(1,4) = INFO(1,4) -1
     IF(NOUT.EQ.1) THEN
          WRITE(2,*)
          WRITE(2,*) ' INPUT AFTER SEXE CHANGES '
          DO 910 IA = 1,2
          WRITE(2,10) (INFO(IA,J),J=1,4)
910       CONTINUE
          DO 900 IA = 1,2
          WRITE(2,10) (STATEC(IA,J),J=1,4),(STATED(IA,J),J=1,4)
900       CONTINUE
          DO 912 IA = 1,2
          WRITE(2,10) (STATEX(IA,J),J=1,4),(STATEY(IA,J),J=1,4)
912       CONTINUE
          DO 914 IA = 1,2
          WRITE(2,10) (STATEP(IA,J),J=1,4)
914       CONTINUE
     END IF
     RETURN
     END
```

SUBROUTINE SPERBUF

```
C
C
C
C
C
      SUBROUTINE SPERBUF(STATEC,STATED,INFO,KROW,KCOL,KILL,STATEX,
      C                        STATEY,STATEP,NOUT)
      INTEGER INFO(2,4)
      INTEGER KROW,KCOL
      INTEGER NOUT
      INTEGER KILL(41,8)
      INTEGER STATEC(30,4),STATEX(30,4)
      INTEGER STATEP(30,4)
      REAL STATED(30,4),STATEY(30,4)
C
C     FORMAT STATEMENTS
10    FORMAT(1X,4I4,1X,10F10.4)
C
C     PLACE DECAYED STIMULUS IN KILL
      KILL(STATEX(KROW,KCOL),6) = KILL(STATEX(KROW,KCOL),6) + 1
      IF(NOUT.EQ.1) THEN
           WRITE(2,*)
           WRITE(2,*) ' KILL DECAY IN PERCEPTION ',STATEX(KROW,KCOL)
      END IF
C
C     BEGIN BUFFER REORGANIZATION
      DO 100 IA = KROW,INFO(1,2)
      STATEC(IA,2) = STATEC(IA+1,2)
      STATED(IA,2) = STATED(IA+1,2)
      STATEX(IA,2) = STATEX(IA+1,2)
      STATEP(IA,2) = STATEP(IA+1,2)
      STATEY(IA,2) = STATEY(IA+1,2)
100   CONTINUE
C
C     DECREMENT PERCEPTION INFO
      INFO(1,2) = INFO(1,2) - 1
C
C     OUTPUT
      IF(NOUT.EQ.1) THEN
           WRITE(2,*)
           WRITE(2,*) ' OUTPUT IN STATE AFTER PERCEPTION BUFFER DECAY '
           DO 210 IA = 1,2
210        WRITE(2,10) (STATEC(IA,J),J=1,4),(STATED(IA,J),J=1,4)
           DO 212 IA = 1,2
212        WRITE(2,10) (STATEX(IA,J),J=1,4),(STATEY(IA,J),J=1,4)
           DO 214 IA = 1,2
214        WRITE(2,10) (STATEP(IA,J),J=1,4)
      END IF
      RETURN
      END
```

SUBROUTINE SRECBUF

```
C
C
C
C
C
      SUBROUTINE SRECBUF(STATEC,STATED,INFO,KROW,KCOL,KILL,STATEX,
     C                   STATEY,STATEP,NOUT)
      INTEGER INFO(2,4)
      INTEGER KROW,KCOL
      INTEGER NOUT
      INTEGER KILL(41,8)
      INTEGER STATEC(30,4),STATEX(30,4)
      INTEGER STATEP(30,4)
      REAL STATED(30,4),STATEY(30,4)
C
C     FORMAT STATEMENTS
10    FORMAT(1X,4I4,1X,10F10.4)
C
C     PLACE DECAYED STIMULUS IN KILL
      JA = STATEX(KROW,KCOL)
      KILL(JA,7) = KILL(JA,7) + 1
      IF(NOUT.EQ.1) THEN
           WRITE(2,*)
           WRITE(2,*) ' KILL DECAY IN RECOG ',STATEX(KROW,KCOL)
      END IF
C
C     BEGINNING OF REORDERING OF RECOGNITION BUFFER
      DO 100 IA = KROW,INFO(1,3)
      STATEC(IA,3) = STATEC(IA+1,3)
      STATED(IA,3) = STATED(IA+1,3)
      STATEX(IA,3) = STATEX(IA+1,3)
      STATEP(IA,3) = STATEP(IA+1,3)
      STATEY(IA,3) = STATEY(IA+1,3)
100   CONTINUE
C
C     DECREMENT RECOGNITION INFO
      INFO(1,3) = INFO(1,3) - 1
C
C     OUTPUT
      IF(NOUT.EQ.1) THEN
           WRITE(2,*)
           WRITE(2,*) ' OUTPUT IN STATE AFTER RECOGNITION BUFFER DECAY '
           DO 210 IA = 1,2
210        WRITE(2,10) (STATEC(IA,J),J=1,4),(STATED(IA,J),J=1,4)
           DO 212 IA = 1,2
212        WRITE(2,10) (STATEX(IA,J),J=1,4),(STATEY(IA,J),J=1,4)
           DO 214 IA = 1,2
214        WRITE(2,10) (STATEP(IA,J),J=1,4)
      END IF
      RETURN
      END
```

SUBROUTINE SEXEBUF

```
C
C
C
C
C
      SUBROUTINE SEXEBUF(STATEC,STATED,INFO,KROW,KCOL,KILL,STATEX,
      C              STATEY,STATEP,NOUT)
      INTEGER INFO(2,4)
      INTEGER KROW,KCOL
      INTEGER NOUT
      INTEGER KILL(41,8)
      INTEGER STATEC(30,4),STATEX(30,4)
      INTEGER STATEP(30,4)
      REAL STATED(30,4),STATEY(30,4)
C
C     FORMAT STATEMENTS
10    FORMAT(1X,4I4,10F10.4)
C
C     PLACE DECAYED STIMULUS IN KILL
      KILL(STATEX(KROW,KCOL),8) = KILL(STATEX(KROW,KCOL),8) + 1
      IF(NOUT.EQ.1) THEN
           WRITE(2,*)
           WRITE(2,*) ' KILL DECAY IN EXECUTION ',STATEX(KROW,KCOL)
      END IF
C
C     BEGIN BUFFER REORGANIZATION
      DO 100 IA = KROW,INFO(1,4)
      STATEC(IA,4) = STATEC(IA+1,4)
      STATED(IA,4) = STATED(IA+1,4)
      STATEX(IA,4) = STATEX(IA+1,4)
      STATEP(IA,4) = STATEP(IA+1,4)
      STATEY(IA,4) = STATEY(IA+1,4)
100   CONTINUE
C
C     DECREMENT EXECUTION INFO
      INFO(1,4) = INFO(1,4) - 1
C
C     OUTPUT
      IF(NOUT.EQ.1) THEN
           WRITE(2,*)
           WRITE(2,*) ' OUTPUT IN STATE AFTER EXECUTION BUFFER DECAY '
           DO 210 IA = 1,2
210        WRITE(2,10) (STATEC(IA,J),J=1,4),(STATED(IA,J),J=1,4)
           DO 212 IA = 1,2
212        WRITE(2,10) (STATEX(IA,J),J=1,4),(STATEY(IA,J),J=1,4)
           DO 214 IA = 1,2
214        WRITE(2,10) (STATEP(IA,J),J=1,4)
      END IF
      RETURN
      END
```

SUBROUTINE SPERERR

```
C
C
C
C
C
      SUBROUTINE SPERERR(JTEMPC,CONMAT,CONFUS,SEED)
      INTEGER JTEMPC
      INTEGER*4 SEED
      INTEGER CONFUS(41,41)
      REAL CONMAT(41,45)
      RA = RAN(SEED)
      DO 100 IA = 1,41
      IF(RA.LE.CONMAT(JTEMPC,IA)) THEN
           CONFUS(JTEMPC,IA) = CONFUS(JTEMPC,IA) + 1
           JTEMPC = IA
           GOTO 110
      END IF
100   CONTINUE
110   CONTINUE
      RETURN
      END
```

```
C
C
C
C
C
      SUBROUTINE SGPM(STIMDUR,INTERC,INTERG,NGPM)
      INTEGER NGPM
      REAL INTERC(15),INTERG(15)
      REAL STIMDUR(43)
      STIMDUR(42) = INTERC(NGPM-5)
      STIMDUR(43) = INTERG(NGPM-5)
      RETURN
      END
C
C
C
C
C
      REAL FUNCTION SGAMMA(ALPHA1,BETA1,DATLAT,SEED,JROW,JCOL,
     C                     JPRO,JCHR)
      INTEGER*4 SEED
      REAL ALPHA,BETA
      REAL ALPHA1(2,4),BETA1(2,4)
      REAL A,B,D,Q,U1,U2,THETA,W,V,Y,Z
      REAL DATLAT(41,45)
      ALPHA = ALPHA1(JROW,JCOL)
      BETA = BETA1(JROW,JCOL)
C     ASSIGNING PERCEPTION TIME: PERCEPTION TIME = OBSERVED CORRECT TIME -
C          RECOGNITION TIME - EXECUTION TIME.  BETA(1,2) (VARIANCE FACTOR)
C          ASSIGNED BEFOREHAND.  ONLY A MATTER OF ASSIGNING ALPHA
C          SO THAT ABOVE EQUAITON HOLDS
      IF((JROW.EQ.1).AND.(JCOL.EQ.2)) THEN
           RSUB = ALPHA1(1,3)*BETA1(1,3) + ALPHA1(1,4)*BETA1(1,4)
           ALPHA = (DATLAT(JCHR,42) - RSUB)/BETA
      END IF
      A = 1./((2.*ALPHA - 1.)**.5)
      B = ALPHA - LOG(4.)
      Q = ALPHA + 1./A
      THETA = 4.5
      D = 1. + LOG(THETA)
10    U1 = RAN(SEED)
      IF(U1.LE.0) GO TO 10
      U2 = RAN(SEED)
      V = A*LOG(U1/(1. - U1))
      Y = ALPHA*EXP(V)
      Z = U1*U1*U2
      W = B + Q*V - Y
      IF((W + D - THETA*Z).GE.0) THEN
           SGAMMA = Y*BETA
           RETURN
      END IF
      IF(W.GE.LOG(Z)) THEN
           SGAMMA = Y*BETA
```

```
     RETURN
ELSE
     GO TO 10
END IF
RETURN
END
```

SUBROUTINE SZERO

```
C
C
C
C
C
      SUBROUTINE SZERO(CONFUS,NCHR,INFO,NCNT,KEEP,KEEPSP,KILL,
      C              STATEC,STATEP,STATEX,STATED,STATEY,TIME,TIMESP,
      C              NPER)
      INTEGER NCHR
      INTEGER NCNT
      INTEGER NPER
      INTEGER CONFUS(41,41)
      INTEGER INFO(2,4)
      INTEGER KEEP(41,46)
      INTEGER KEEPSP(5)
      INTEGER KILL(41,8)
      INTEGER STATEC(30,4)
      INTEGER STATEP(30,4)
      INTEGER STATEX(30,4)
      REAL STATED(30,4)
      REAL STATEY(30,4)
      REAL TIME(41,45)
      REAL TIMESP(5)
C ----  NPER
      NPER = 0
C ----  INFO
      INFO(1,1) = 1
      INFO(2,1) = NCNT
C ----  CONFUS
      DO 100 IA = 1,NCHR
      DO 100 IB = 1,NCHR
      CONFUS(IA,IB) = 0
100   CONTINUE
C ----  KEEP
      DO 105 IA = 1,NCHR
      DO 105 IB = 1,45
      KEEP(IA,IB) = 0
105   CONTINUE
C ----  KEEPSP
      DO 110 IA = 1,5
      KEEPSP(IA) = 0
110   CONTINUE
C ----  KILL
      DO 120 IA = 1,NCHR
      DO 120 IB = 1,8
      KILL(IA,IB) = 0
120   CONTINUE
C ----  STATEC,STATEP,STATEX
      DO 130 IA = 1,30
      DO 130 IB = 1,4
      STATEC(IA,IB) = 0
      STATEP(IA,IB) = 0
      STATEX(IA,IB) = 0
```

```
130  CONTINUE
C    IN ORDER TO START SIMULATION, WANT FIRST CHARACTER IN ARRIVAL TO
C         BE A BLANK, 42 OR 43 IS OK
     STATEC(1,1) = 42
     STATEX(1,1) = 42
C ----  STATED,STATEY
     DO 140 IA = 1,30
     DO 140 IB = 1,4
     STATED(IA,IB) = 0.
     STATEY(IA,IB) = 0.
140  CONTINUE
C    IN ORDER TO START SIMULATION, NEED TO ASSIGN A DURATION TO BLANK
C         SO THAT SMINI DOESN'T THINK EVERYTHING IS FINISHED
     STATED(1,1) = 50.
     STATEY(1,1) = 50.
C ----  TIME
     DO 150 IA = 1,NCHR
     DO 150 IB = 1,45
     TIME(IA,IB) = 0.
150  CONTINUE
C ----  TIMESP
     DO 160 IA = 1,5
     TIMESP(IA) = 0.
160  CONTINUE
     RETURN
     END
```